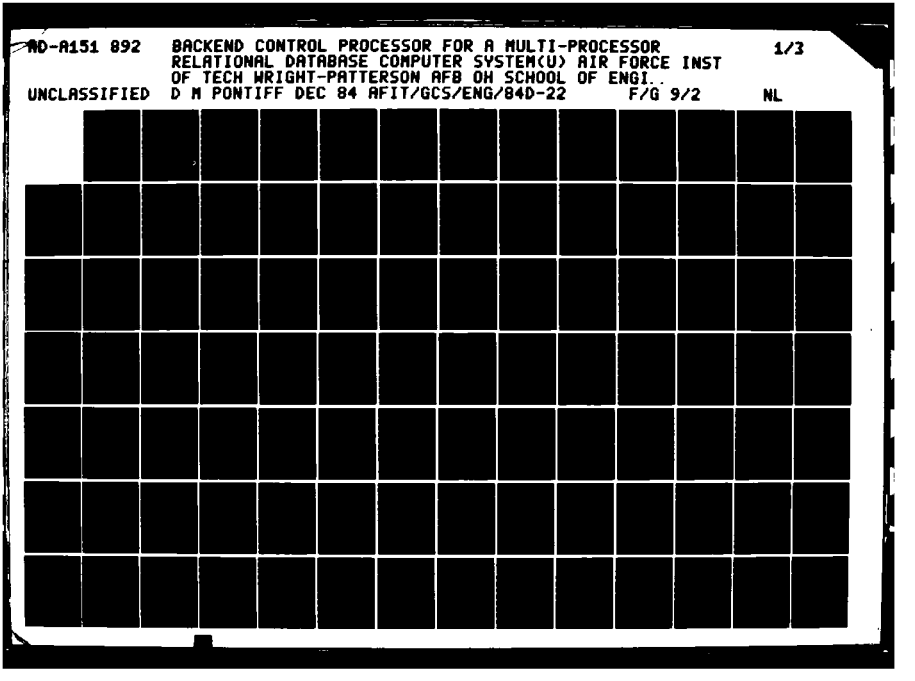
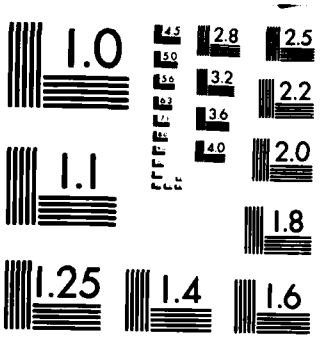
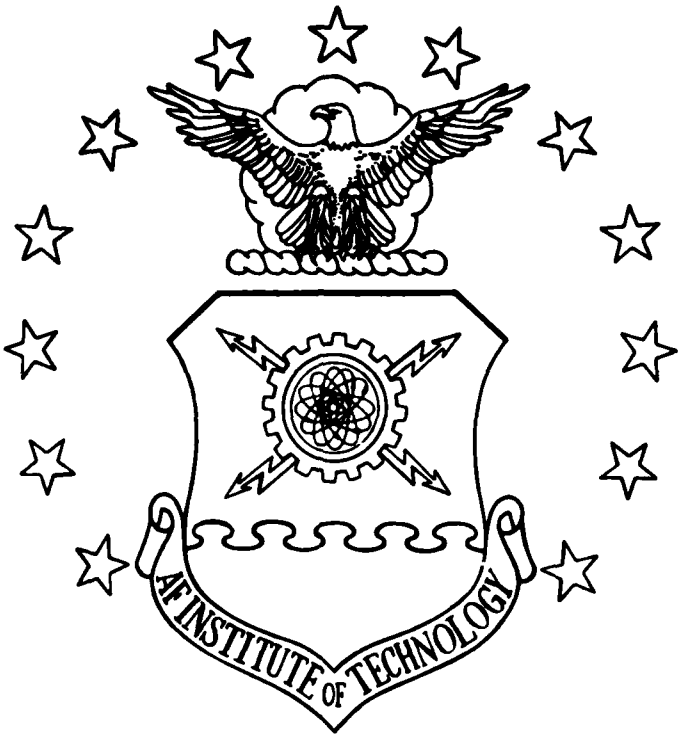
AD-A151 892 BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/3

UNCLASSIFIED D M PONTIFF DEC 84 AFIT/GCS/ENG/84D-22 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM

THESIS

Dale M. Pontiff
Captain, USAF

AFIT/GCS/ENG/84D-22

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

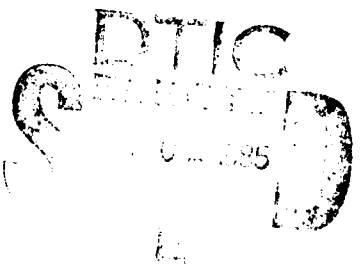
AFIT/GCS/ENG/84D-22

BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM

THESIS

Dale M. Pontiff
Captain, USAF

AFIT/GCS/ENG/84D-22



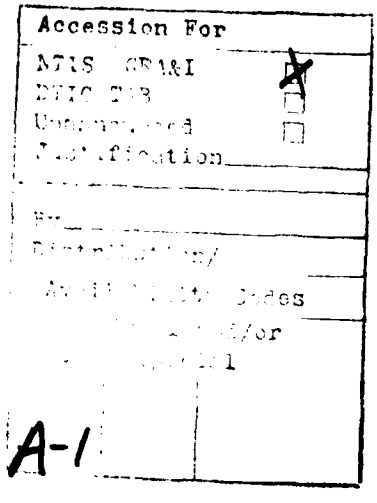
Approved for public release; distribution unlimited

AFIT/GCS/ENG/84D-22

BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Computer Information



Dale M. Pontiff
Captain, USAF

December 1984

Approved for public release; distribution unlimited

Preface

In 1981, Robert Fonden began the design and development of a Backend Multi-Processor Relational Database Computer System. This thesis addresses a single component of this system, the Backend Control Processor (BCP). The BCP is responsible for managing the actions of the entire backend system. It must provide direction for many slave microprocessors, and control the system paging algorithms. This paper covers, in detail, the design of the BCP for Robert Fonden's Backend architecture.

I would like to give special thanks to my thesis advisor Dr. Thomas C. Hartrum for his guidance, patience, and encouragement during the development of this thesis. Each time the effort grinded to a halt, he provided the necessary boost to set my mind in motion once again. Thanks are also due to Dr. Henry Potoczny for his assistance, especially in Chapter 4, and to the lab technicians, Dan Zambon, Charlie Powers, and Capt. Lee Baker for their prompt, courteous responses to all of my questions and problems.

Finally, I wish to thank my wife, Beth, for all her help, support, and encouragement during my graduate studies.

Contents

List of Figures

Abstract

This paper discusses the design and developement of a control processor for a multi-processor relational database machine. The objective was to create the software needed to allow a micro-processor to receive relational query trees from a frontend processor, and to distribute the work load between several other slave processors.

The requirement analysis of the controller determined that the controller must provide three major functions within the backend database machine. It must assign slave processors to query operations, control the system paging, and manage file creation and deletion. Next, the thesis proves that each query operation can sucessfully be split across several slave processors and the results be recombined to provide the same response as a query executed on a single processor. Finally, the thesis gives a detailed description of the software algorithms use by the BCP to manage the backend system.

BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM

I Introduction

Background

In recent years, there has been a growing requirement for "easy-to-use" databases. This growth is caused by an increase in the number of people without computer programming background relying on computers to organize information for their jobs. Because of the "easy-to-use" requirement, relational databases have been growing in number and popularity.

Relational databases are based on solid mathematical principles, using relational algebra, yet their use is intuitive. At the user level, relational databases provide a tableau view of the data. This view makes it easier to train people in the use of the database.

While relational databases solve the "easy-to-use" criterion, there is still the problem of handling the substantially increasing amounts of information processing. One of the major drawbacks of relational databases is that they tend to be slower than other database models when dealing with large databases. As the database grows in size and complexity, the computer resources will become saturated, and system degradation will begin.

There are currently many research efforts underway to

improve computer databases. Within AFIT alone, there is work being done on several theses concerning four major areas of database development. This interest in databases at AFIT is largely directed by Dr. Thomas Hartrum. These four areas of research include (7):

* intelligent software (reduce paging)
* specialized architecture (provide parallel processing)
* distributive DBMS (allow several DBMSs on many systems to act as one)
* user interfaces (provide user-friendly tools and interfaces to simplify complexity of the system)

As knowledge is gained in each of these areas, it is possible to combine two or more of the features to provide an extremely powerful DBMS.

This thesis deals with providing a specialized computer architecture to allow parallel processing on a given query. This project began with Robert Fonden's design of a multiprocessor DBMS in 1981 (6). Further design work on the system was done by William Rogers in 1982 (12).

The Fonden architecture makes use of a multiple instruction multiple data stream (MIMD) backend database architecture to improve system throughput. A MIMD system allows a high degree of parallel processing to be achieved by separate functional units that perform operations simultaneously. To do this, the data has to be distributed among the multiple functional units.

With the introduction of inexpensive microsystems, it has become feasible to use a number of secondary computers as

a multifunctional backend system. By placing the database on a backend, the host system is relieved of many of the time consuming phases of database processing. This approach adds more computing power to the existing host system without a major upgrading to a newer, more powerful machine.

With a multi-processor backend, one can take advantage of the three types of parallelism inherent in relational database queries to improve service time. The three types of parallelism are:

* independent parallelism
* node splitting
* pipelining

Independent parallelism is simultaneously processing two (or more) parts of a query which will be joined at a later stage in the query.

Node splitting is having several processors simultaneously act on different pages in the same query step.

Pipelining is having the output of a process(es) being immediately fed into a second processor(s) to complete the next step of the query.

Fonden's thesis has described a computer architecture in which a backend system with many small processors can be used to enhance the performance of a relational database system (6: 68-91). This architecture uses a master/slave relationship in which a Backend Control Processor (BCP) controls the action of many Query Processors (QPs). His system consists

of (See Figure 1):

* a mass storage device
* a BCP
* eight QPs
* six Internal Memory Modules (IMM) per QP

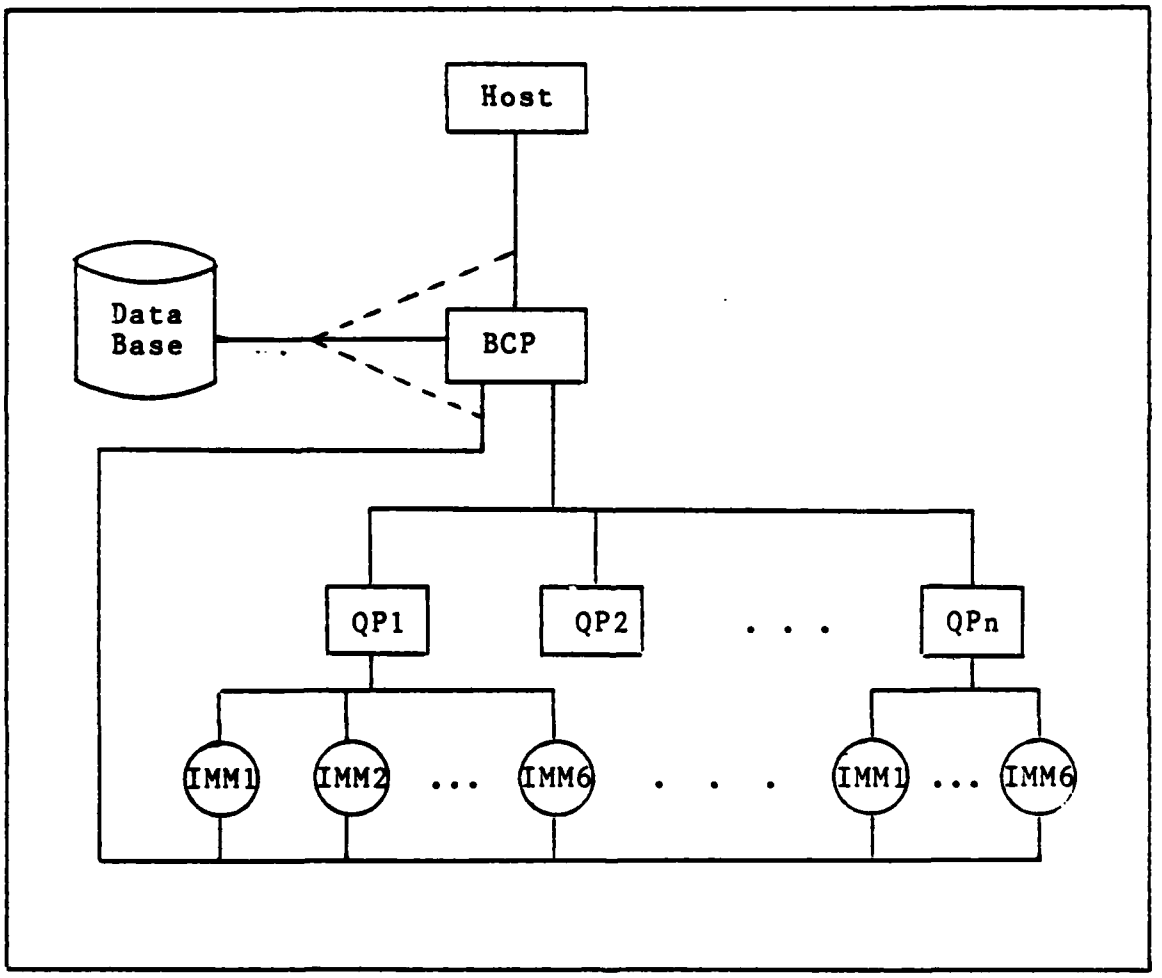


Figure 1. Fonden's Original Physical Design Approach
Source: (6: 75).

This architecture makes use of both the backend system and the inherent parallelism of relational queries to speed up the retrieval/update time of the database.

As development progressed, a slight modification was made to Fonden's architecture. A frontend processor has been added to the system. The frontend processor could be connected to a local network, a host system, or several user terminals. This will hopefully improve the system's modularity, flexibility and performance capacity. The frontend would be responsible for query optimization, transaction log, and security checks of the database.

Statement of the Problem

The purpose of this thesis is to provide a working model of the Backend Control Processor (BCP) for Fonden's Backend Database Computer System. At the initial stages of development, emphasis is placed on ease of understanding, portability, and maintainance rather than on speed and efficiency. The reason for this is because this is an on-going thesis project and other students will have to pick up the system at a later date. It is hoped that the multiprocessors alone will give the increased performance desired, and code optimization may occur at a later time.

Scope

Upon the completion of this project, the BCP should be able to:

* receive a query message from the frontend
* allocate the necessary QPs to a query
* communicate with each QP by passing "query steps" and other necessary information
* receive communication from the QPs upon the completion of a page or "query step"
* page relations into and out of each QP's IMMs
* maintain bookkeeping information about the DB
* transmit a response back to the frontend

Due to the time constraints, the following areas will not be addressed in this thesis:

* optimization of queries
* optimization of selecting types of parallel processing
* selecting the optimum number of QPs for a query
* transaction log
* security
* backup

NOTE: Whereever possible, hookups for the above items will be included in the design of the BCP.

Current Knowledge

Fonden has completed a feasibility study on the overall project (6: 67) and provided long term requirements and goals (6: 68-73). Rogers has completed the analysis and high-level design of the Query Processors (12).

Approach

The project began with preliminary study on two developing backend database architectures. This was to aid in an overall understanding of the project, and hopefully point out some pitfalls to avoid. Next, a thorough analysis of the BCP was performed with the use of Structure Analysis and Design Technique (SADT) (11: 62-69). Following the analysis, the system was implemented in a modular style using the 'C'

language on the S-100 multi-processor system. The final step was to test the system to determine what problems still exist.

Overview of the Thesis

This thesis is divided into 8 chapters.

Chapter 1 is an introduction to the thesis. It gives a brief background discussion of the backend system, and defines the purpose of the thesis.

Chapter 2 discusses the preliminary study phase of the thesis and insights gained by the author through the review of existing backend database architectures.

Chapter 3 discusses particular problems encountered during the BCP requirements analysis, followed by an overall view of the requirements of each component of the backend system. Finally, it gives a brief summary of the different query step operations and BCP commands.

Chapter 4 discusses paging theory related to the backend system, and provides proofs that the query steps may be correctly split across several different query processors.

Chapter 5 discusses the local hardware configuration for the initial versions of the backend system.

Chapter 6 discusses the data structures used throughout the BCP software.

Chapter 7 discusses the algorithms for assigning query processors to a query step, and providing system paging.

Chapter 8 provides a summary of the thesis, and suggested areas where the BCP and backend system can be improved.

Material and Equipment

Implementation and testing for this project was done on the S-100 system available in the computer lab. This system consists of a Super-Quad board by the Advanced Digital Corporation (1: 1) with 64K memory, and dual 8 inch disk drives. The operating system was CP/M version 2.2. All code was written in BDS C (15).

II Preliminary Study

Introduction

The preliminary study concentrated mainly on two other multi-processor database systems. The first one is being developed at the Ohio State University (OSU) under a contract with the Office of Naval Research. The second database system (DIRECT) is operational at the University of Wisconsin, but still being enhanced.

The purpose of the preliminary study was to gain some knowledge of other database machines with specialized architecture. This could possibly give some insight into the design and configuration of related existing systems. The study achieved its goal by bringing to the surface several problems (and solutions) which were not previously considered.

Ohio State University's Database Efforts

The OSU database employs many micro-processors, each with its own disk drive. The data is evenly distributed across each processor, and the query is simultaneously broadcast to all processors. The results are then returned to the controller.

The OSU technical report (8) gives a detailed report on the design considerations important to their backend database system. Not only does it discuss the development of their

own Multi-Backend Database System, but it also includes an excellent summary of the advantages and weaknesses of several other backend systems. These are:

* RDBM - a relational DB system
* DIRECT - a multiple backend relational system
* Stonebraker's Machine - a "distributed" DB system
* DBMAC - an Italian DB system

The report discusses several key database problems which the author had not initially considered. The placement and accessiblity problem of the system data dictionary became apparent (See Chapter 3) during this review. The report also caused a more concrete consideration of the degree of concurrency and the locking mechanisms of the Backend System.

One of the interesting features of the OSU System is that it is being developed to allow linear performance growth proportional to the number of backends employed. This differs from most systems, where the performance of each processor decreases with the addition of each new processor until the additional overhead/cost out weights the performance gains.

The notion of linear performance growth is interesting, and several days were spent trying to incorporate this feature into Fonden's Backend System. It was soon discovered that this was beyond the capability of the system because to achieve this goal would require the removal of any decision making algorithms from the controller, and a way had to be found to eliminate I/O contention between the processors.

Because Fonden's architecture does not have a linear

performance growth with the addition of each QP, it is conceivable that the backend system could become saturated. Upon realizing that the Fonden system had an upper bound, while OSU's system could grow (theoretically) forever, the author considered abandoning the thesis because it would soon be obsolete.

In the real-world, Fonden's system should be able to handle a reasonably large number of QPs before becoming saturated, and is capable of surviving the loss of a QP, whereas on the OSU system the loss of a QP also means the loss of part of the database. Thus, any output relations created on a partial system would possibly be incorrect. So, while Fonden's architecture has an upper bound on its growth, it is less vulnerable to hardware failure and does indeed have a future in database management.

DIRECT

The second system studied was the DIRECT (3) (5) system of the University of Wisconsin. This was choosen because it is very similar in design to Fonden's system. Both systems use a Controller Processor to supervise many micro-processors that share a commom memory unit. The major difference between the systems is the location of the database files. In Fonden's design, the backend system has its own dedicated storage device (accessable only by the backend). In the DIRECT architecture, it appears as if the database files are

stored on the host system's disk drives, thus the host system must perform the actual paging of data into and out of the backend memory unit.

The Controller in the DIRECT system performs the following major functions:

* creation/deletion of relations
* packet (task) assignments to slave processors
* memory management

These are essentially the same major functions provided by the BCP in Fonden's System.

The review of the DIRECT system did not turn up any additional problems not already discovered, but did mention some of the same problems. It also mentioned that the handling of interprocess messages was one of the most complex areas within their system.

Summary

The reveiw of other systems stressed the importance of hardware restrictions, but neither dicussed detailed methods for recombining the output of several processors into a single result relation. The preliminary study made the author aware that data paging and bus contention would be major problems to any further design, and that special attention should be given in these areas.

III Functional Requirements Analysis of the Backend Control Processor

Introduction

This chapter will discuss specific problems encountered while performing the functional requirement analysis of the BCP. Briefly, it covers:

* the addition of a frontend to the Fonden architecture
* the functional breakdown of a conventional DBMS
* the major responsibilities of each major component in the backend relational database computer system
* the location and accessability of the Data Dictionary
* handling of update type queries
* modification made to Roger's Query Processor analysis
* summary of query step operations

Appendex C contains the Structure Analysis and Design Technique (SADT) and the Data Dictionary developed during this study.

Addition of Frontend

Upon beginning the functional requirement analysis for the Backend Control Processor (BCP), several major problems began to arise. One of the foremost was the fact that Fonden's architecture required a very tight coupling with the host system. He assumed that the host would be passing down an optimized query tree to the backend database system.

There are several problems with this structure. First is the fact that it defeats one of the reasons for using a backend computer; that is modularity, the ability to easily replace the host system and still have an intact DBMS. Any

new system would require programming to enable it to pass a complex tree structure rather than a simple query message.

The second major problem is the location of the database data dictionary. The data dictionary should be placed with the database, but the host system needs access to it to be able to optimize the query. This means that the host system must be able to access the backend's mass storage device, or the backend must download the entire data dictionary to the host system. Both options have undesirable side effects.

In an attempt to isolate the backend system from the host system as much as possible, a frontend processor was added to the architecture. Its initial function was the optimization of queries, but as the functional requirements of the system were studied, other functions were assigned to the frontend.

In the modified architecture (See Figure 2), the Frontend (FE) is connected to the host system. Incoming queries and commands are validated (and optimized) by the FE. The FE must access the database data dictionary to be able to perform these functions, so it is directly linked to the Mass Storage Unit (MSU). The numerous IMMs of Fonden's architecture are replaced by the Memory Buffer Unit (MBU), but it still serves the same basic function; provide a fast scratch pad for the QPs. The BCP does not need to directly access data on the MSU, but it must be able to control the system paging and file creation/deletion process.

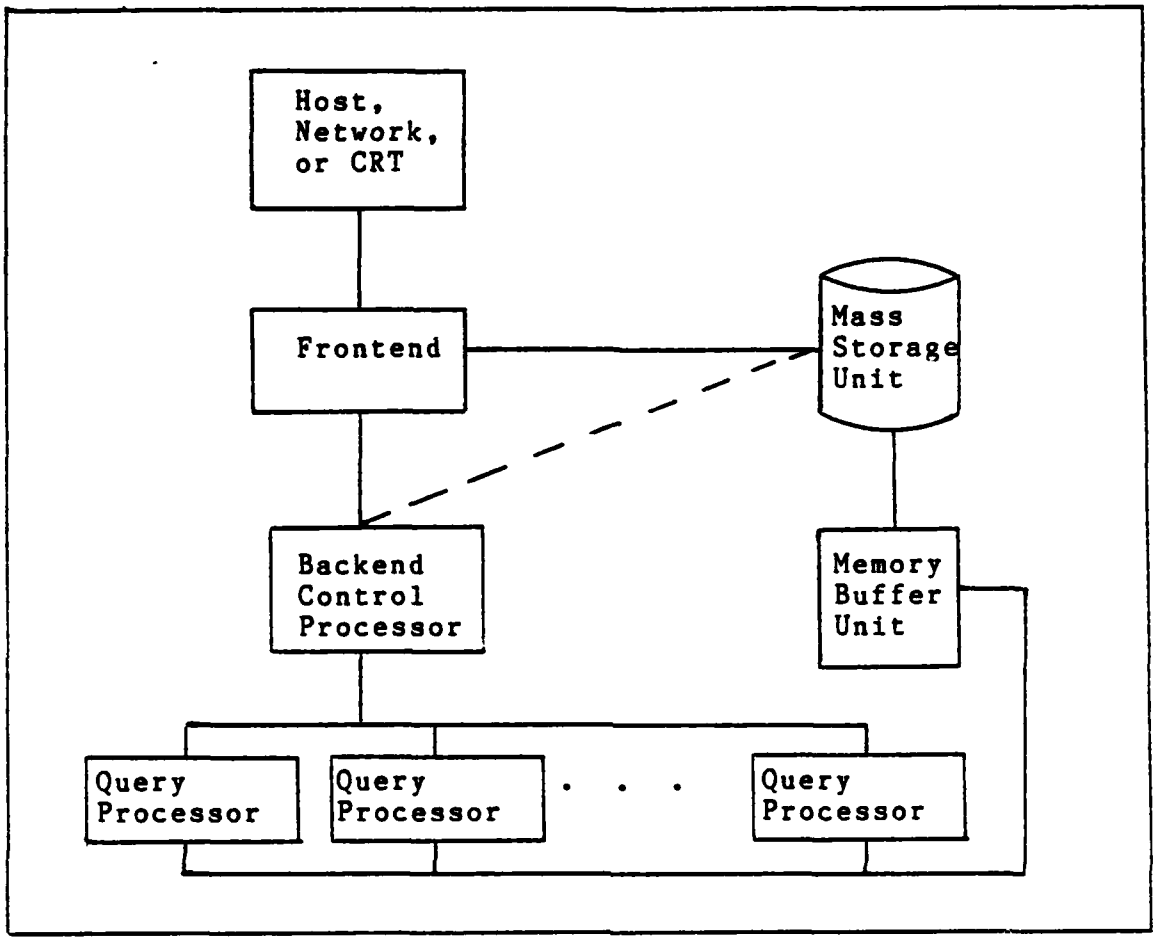


Figure 2. Modified Physical Design of Backend System.

Functional Breakdown of a DBMS

With the addition of the frontend (FE), it became necessary to reevaluate the entire functional relation of each component in the modified backend system. Several unsuccessful attempts were made to determine the operations of each component, and it became apparent that a new mode of attack was needed to help solve this problem.

It was decided that a functional breakdown of a conven-

tional single processor DBMS would reveal the logical modules of the system (See Appendix B). This breakdown was completed only at the highest level to serve as a guide to enable a modular breakdown on the multi-processor system. The major functions of a DBMS are:

* get query
* analyze syntax
* verify access
* log transaction
* optimize query
* execute query
* manage Data Dictionary
* send answer

After succussfully breaking the DBMS into its functional units, it is a simple matter to map the functional operations into the hardware components of the Backend Relation Database Computer System.

On a large database, the "execute query" step is the time consuming process, and it is this step that the multi-processor architecture is aimed at reducing.

Frontend's Responsibilities

The FE's primary job is to optimize the query. To do this, it must first receive the query and analyze the syntax. Since it is wise to catch errors at the earliest opportunity, access verification should be done prior to optimization. It also seems prudent to log the transaction in its original form (rather than as a complex query tree). Given these guidelines, it is reasonable to place all of these DBMS functional operations on the FE. Because the FE must communi-

cate with the host system to receive the query, it is logical to have the FE send the results back to the host. Last, the FE is the only processor that requires access to the Data Dictionary. Therefore, it must also manage the Database Data Dictionary. Given these requirements, the FE must:

* communicate with the outside world (receive query and send replies)
* communicate with the BCP (pass down optimized queries and receive responses)
* communicate with the Mass Storage Unit (MSU) (read/write file; delete files)
* analyze syntax of a query
* verify access rights of a query
* log transactions
* optimize the query
* manage the DB Data Dictionary

In fact, the FE has become a single processor DBMS except for the "execute query" function.

Backend Control Processor's Responsibility

The BCP's primary job is to provide control of many QPs. This is largely unchanged from Fonden's original proposal (6: 109-113), except that the BCP can dispense with query validation, and now the BCP talks to the FE instead of the host system. In light of the modified architecture, the BCP must:

* communicate with the FE (receive queries and send responses)
* control the MSU (tell it what and where to read/write into the MBU; create and delete files)
* control the QPs (tell them what step to perform; where to read/write data in the MBU)
* manage QP allocation
* manage system paging
* manage creation/deletion of temporary relations
* provide job control

Query Processor's Responsibilities

The QP's primary job is to provide relational operations on a page of a relation. This is unchanged from Fonden's proposal, although minor modifications were made to Rogers' analysis. The QPs are not interested in where the data comes from or where the output goes. They simply perform a specific relational operation on a specific page in the MBU, and store the results in another page within the MBU. The QPs must be able to:

* read/write into the MBU
* communicate with the BCP (receive query steps and paging information)
* perform the necessary relational operations

Mass Store Unit's Responsibilities

The MSU's primary job is to provide permanent storage for the database, and provide a file control mechanism (ie. the ability to create, delete, and append to a file). It must be able to:

* read/write into the MBU
* communicate with the FE (receive/send files and DD)
* communicate with the BCP (receive directions about where to read/write in the MBU; receive command on creation or deletion of files)
* provide file control commands (create/delete/append)
* provide a permanent storage media

Memory Buffer Unit's Responsibilities

The MBU's primary job is to provide a fast buffer storage for the QPs. Ideally, any QP could access any page in the MBU allowing pipelined operation and shared pages. The MBU is what Fonden described as the IMMs in his proposal, but

the paging buffers should be thought of as a single component rather than as many small separate units, thus the change in names. It must:

* allow the MSU to read/write to any buffer page
* allow the QP to read/write to any buffer page
* provide approximately 8 pages (or more) of buffering per QP

Location and Accessibility of the Database Data Dictionary (DD)

One of the major concerns of splitting the database manager across multiple processors was where to locate the Data Dictionary for the database. After much discussion and study, it was decided that by including certain critical data in each query step, only the FE processor would require access to the DD. Hence the DD is placed on the FE, where the most critical need exists.

There are obvious advantages in having only one of the processors accessing the DD. These include not having to provide a locking scheme on the DD, reducing processor communications (to access the DD), reducing storage requirements on the other processors, and eliminating the need to propagate changes in the DD to multiple processors. The drawbacks are requiring the FE to do more bookkeeping during the query tree build, and increasing the size of the query steps being passed between processors.

Since the FE has already insured that domain boundaries are not crossed (e.g. compare 'city' with 'num_of_workers'), and that illegal actions on an attribute are not performed

(e.g. sum a character field), the BCP and QPs are not required to make these checks and do not need most of the information stored in the DD. The BCP simply needs the name of all DB files to be accessed in the query (which it would require regardless of the DD location), and the size (in pages) of each file. This is because the BCP's function is largely to provide paging control of the files, which does not require knowledge about the contents of the file. The QPs need the starting address and length of each attribute field accessed (12: 39-69) (if working on fixed records), or just the attribute field number (if working on varying records). This is because the QPs simply need to know where to stop and start in comparison/modifications of a field, and are not concerned with the contents of data in a field.

As mentioned above, this requires additional bookkeeping by the FE while building the query tree, and is implementation-dependent (a detailed account is given in Appendix D).

Handling Update Type Queries

On the first attempt at the functional analysis of the BCP, file updates and file retrievals were separated into two different functional areas. The reason for this was because Roth's Query Optimizer was designed with retrievals in mind, but provides little directions for updates. On subsequent reviews in this area, the discussion of whether retrievals and updates should be treated as a single function or as two

separate functions was questioned. It was decided that the decision should be based on functional requirements and not on existing tools (although during implementation the reverse may be true). Further study determined that the paging requirements were basically the same and there were no major differences between the two types of operations from the BCP view.

This means that Roth's Query Optimizer (13: 51-78) must be modified to handle updates as well as retrieval requests, or that updates must be performed in the form they are entered. Because an update request may only modify a single relation, it will generally consist of only a single node and will not require optimization. Therefore, there should be no problem in placing the update requests into a tree form.

Modification to Rogers' Query Processor Analysis

While developing the paging algorithms for the BCP to handle the different relational operations, it became apparent that changes would be necessary in the design of the QPs. Rogers discussed the relational operations he believed were essential on the QPs for the Backend System to operate (12: 36-40). The following changes are needed to the QP to provide the support the BCP requires:

* "average" is replaced by a "sum" operation
* "compress" is replaced by "sort" and "union" operations
* modify "insert" with a union type operation which reports dupilicate keys
* "intersect" is added

* modify "min/max" operation to return a value, not a tuple
* modify value-returning operations to provide the answer in the response message, not as an output relation

A brief justification for these changes is given below, with detailed paging requirements discussed in a following chapter.

Replace Average.

The reason for eliminating the "average" operation from the QP operations is because the "average" requires the QP to maintain information across the entire relation, not just a single page at a time. It is impossible for the QP to take the average of each individual page and produce an overall average for the relation. This means that either the "average" operation must be removed from the Data Manipulation Language (DML) or an alternative method must be found to provide it. Two workable alternatives are discussed:

* a weighted average
* a sum operation divided by the count

In the final product, a weighted average should be implemented, but for the current system, a "sum" operation shall be added to the QP. The BCP will have to compute the average by dividing the sum by the tuple count.

Replace Compress.

The "compress" operation is removed for the same reason as the "average" operation; a reasonable paging algorithm does not exist. In its place are two operations; "sort" and

"union". The reason behind this is that it simplifies the removal of duplicate tuples (keys). To remove duplicates, each tuple must be compared against every other tuple in the relation. By sorting the data, duplicate tuples will be adjacent to each other and can easily be removed during the sort phase. The reason both a "sort" and "union" operation are required is because a QP cannot sort the entire relation at once. It can only sort the portions of the relation which are in memory. Therefore, each QP can sort a portion of the relation, and then each of these parts can be unioned together with a merge sort algorithm.

Modify Insert.

The "insert" operation discribed by Rogers (12: 52-53) is essentially a "union" operation of a single tuple into an existing relation with an error reporting capability for unwanted duplicate keys. The "insert" operation should be expanded to allow the insertion of an entire relation. This way, the user would be able to insert multiple new tuples into an existing relation with a single query. The reason for this modification is to reduce the cost of inserting tuples, but because duplicate keys must be reported, the modified "insert" operation requires two output files (the output relation, and a duplicate key error file). This changes the ideal page ratio of the MBU per QP from six to eight.

Add Intersect.

An "intersect" operation is added to the QPs because it is easy to implement, and a powerful operation. An "intersect" is a subfunction of a "join". It requires both relations to be identical in attribute types and order, and selects only tuples where all fields are equal. While an "intersect" could be achieved through the "union" and "difference" operations, it is simpler to include it as an individual operation.

Modify Min/Max.

Rogers states that the "minimum/maximum" operations return the tuple with the smallest/largest value. It does not state what happens if many tuples have the same min/max value. Since most other relational system return a value rather than a min/max relation, the "min/max" operation will be modified to return the smallest/largest value of the relation, not the tuple(s).

Modify Value-Returning Operations.

Rogers thought that the QPs should not pass "small data items" directly to the BCP because it is unfeasible for the BCP to consolidate the answer of several QPs. The reverse is actually true. It is easier (and faster) to consolidate single value answers from several QPs than it is to provide the additional paging scheme needed to make a second pass over the resulting output relations. Therefore, operations

which provide a single answer value (such as "count", "sum", "min", and "max") will return the answer in the response message, and not in a output relation.

Summary of Query Step Operations

The initial version of the backend system will have 15 different query step operations. A brief discussion of each operation and its classification is given here. For a more detailed discription, reference Ullman (14: 152-156) and Rogers (12: 39-69). Each operation is classified as either a retrieval or update type operation. Retrievals may be performed on temporary and/or base (permanent) relations, and any output relation will be a temporary relation. Updates may only be performed on base relations, and their output relations replace the old base relations.

The first group consists of "unary" relation operations, and the second group consists of "binary" relation operations. The following definitions are used in describing the operations.

* selection criterion -- A set of boolean (ANDs and ORs) conditions to allow comparison of an attribute field value against a constant or different attribute field value.

* attribute list -- A list of attribute field identifiers.

* modification list -- A list of attribute field identifiers followed by a new value to be stored in the field.

Unary Input Relations

Select (retrieval) -- Given a selection criterion, copy any matching tuples into the output relation.

Delete (update) -- Given a selection criterion, copy any tuple not matching the criterion into the output relation (remove those that do match).

Modify (update) -- Given a selection criterion and a modification list, modify the specific attribute fields of any matching tuple, and store in the output relation. Otherwise, simply copy the unchanged tuple into the output relation. Note, the modification list cannot contain key fields.

Count (retrieval) -- Given a selection criterion, increment a counter for each matching tuple. Upon encountering EOF return the total count to the user.

Project (retrieval) -- Given an attribute list, reorder the tuple's field value to match the new attribute list and remove any fields not listed. Write the modified tuple to the output relation.

Min (retrieval) -- Given an attribute list, perform a project, except instead of writing the modified tuple, compare it against the currently smallest tuple found, keeping the smaller tuple. Upon encountering EOF return the smallest tuple value.

Max (retrieval) -- Same as "min" except maintain the largest value.

Sum (retrieval) -- Given an attribute list, maintain a sum of each attribute field. Upon encountering EOF, return a tuple containing the sums in the order of the attribute list. It is an error to attempt to sum a character field.

Sort (retrieval) -- Given an attribute list, sort the relation based on the order of the attribute list. If no attribute list is provided, sort the relation on its current order.

Binary Input Relations

Product (retrieval) -- Provide the cross product of two relations by concatenating each record in the second relation to the end of each record in the first relation.

Join (retrieval) -- Given a selection criterion, perform a combined "select" and "product" to form an output relation.

Intersect (retrieval) -- Perform an intersect operation on two relations with identical attribute lists.

Diff (retrieval) -- Perform a difference operation on two relations with identical attribute lists.

Union (retrieval) -- If the output relation needs to be sorted and/or duplicate keys must be removed, perform a merge sort algorithm. Otherwise perform a file concatenation. This operation is only valid on two relations with identical attribute lists.

Insert (update) -- Take a base relation and an "insert" file and perform a "union" type operation generating a new base relation and a duplicate key error file.

Summary of BCP Commands

The BCP commands are provided to allow some external control of the BCP's actions. They essentially provide the user the capibility to break out of the query. Any command to the BCP is verified and validated by the FE, and will only allow a user to affect his own queries. The initial set of BCP commands are:

* stop query
* start query
* abort query
* change priority
* status

The "stop query" command will allow a user to halt any of his queries currently in the system. The command causes a specific query to be removed from the active task pool and placed into the inactive task pool. Any QPs working on this query will be preempted by other active tasks.

The "start query" command is used to restart an inactive

query. It moves a specific query from the inactive task pool to the active task pool.

The "abort query" command deletes the query from the backend system. Any QPs currently working on the query are preempted.

The "change priority" command allows a user to change the priority of a specific query.

The "status" command returns the current status of a query. This would include its location in the queue, number of QPs acting against the query, number of query steps left to process, and current task pool (active/inactive).

IV Paging Theory

Introduction

Since the backend system intends to break a query down into query steps, and then further split the query steps between "independent" query processors, it would be nice to insure that the results are the same as if the work were performed on only a single processor. Since there are many references on query optimization (13: 51-78) (14: 268-283) which deal with the breakdown and reordering of individual query steps, this part of the proof will be dispensed with. The chapter will proceed with confidence that the query can correctly be formed into an optimized query tree.

It must proven that all the query processor operations can be split among individual query processors and rejoined to provide the same results. Those operations are:

* select
* delete
* modify
* count
* project
* min
* max
* sum
* sort
* product
* join
* intersect
* difference
* union
* insert

In all the following proofs, R is defined to be an

arbitrary relation, which can be split into disjunct subrelations R1, R2, R3, ... Rn such that $R = \bigcup_{i=1}^{n} Ri$.

S is defined to be an arbitrary relation which can be split into disjunct subrelations S1, S2, S3, ... Sm, such that

$S = \bigcup_{i=1}^{m} Si$.

Select

$$\text{Select}(R) = \bigcup_{i=1}^{n} \text{Select}(Ri)$$

Proof: Show that

(1) Left Hand Side (LHS) $\subset$ Right Hand Side (RHS), and

(2) that RHS $\subset$ LHS.

(1) Let $x \in$ LHS. So that $x \in$ Select (R), thus $\exists$ r $\in$ R such that $x = \text{Select}(r)$. Since $R = \bigcup_{i=1}^{n} Ri$, and $r \in R$, then for some j between 1 and n, $r \in Rj$. Thus Select (r) $\in$ Select (Rj).

Since Select (Rj) $\subset \bigcup_{i=1}^{n}$ (Select (Ri)), then

$x = \text{Select}(r) \in \bigcup_{i=1}^{n}$ (Select (Ri)). Therefore the LHS $\subset$ RHS.

(2) Let $x \in$ RHS. So that $x \in \bigcup_{i=1}^{n}$ Select (Ri), then for some j between 1 and n, $x \in$ Select (Rj), thus $\exists$ r $\in$ Rj such that $x = \text{Select}(r)$. Since $r \in Rj$, then $r \in \bigcup_{i=1}^{n} Ri$, so that

$x = \text{Select } (r) \in \text{Select } (\overset{n}{\underset{i=1}{U}} Ri) = \text{Select } (R)$. Thus RHS $\subset$ LHS.

Since the LHS $\subset$ RHS and the RHS $\subset$ LHS, they are equivalent.

Delete

$$\text{Delete } (R) = \overset{n}{\underset{i=1}{U}} \text{Delete } (Ri).$$

Since the "delete" operation is essentiallly a "select" operation with the selection criterion negated, its proof is the same as that of the "select".

Modify

$$\text{Modify } (R) = \overset{n}{\underset{i=1}{U}} \text{Modify } (Ri).$$

The "modify" operation uses a "select" criterion to determine which tuples to modify. The modification of a tuple is independent (since keys cannot be modified) of the selection order or modifications of other tuples within the relation. Hence, its proof is the same as that of the "select".

Count

$$\text{Count } (R) = \sum_{i=1}^{n} \text{Count } (Ri).$$

Count also uses a "select" criterion to determine whether or not to count a tuple. Since addition is associative and commutative, the grouping and order for tallying matching tuples is irrelevent.

Project

$$\text{Project}(R) = \bigcup_{i=1}^{n} \text{Project}(Ri)$$

Proof: Show that (1) LHS $\subset$ RHS, and (2) RHS $\subset$ LHS.

(1) Let $x \in$ LHS. So that $x \in$ Project (R), thus $\exists$ $r \in R$ such that $x = \text{Project}(r)$. Since $R = \bigcup_{i=1}^{n} Ri$, and $r \in R$, then for some j between 1 and n, $r \in Rj$. Thus $\text{Project}(r) \in \text{Project}(Rj)$. Since $\text{Project}(Rj) \subset \bigcup_{i=1}^{n} \text{Project}(Ri)$, then

$x = \text{Project}(r) \in \bigcup_{i=1}^{n} \text{Project}(Ri)$. Therefore LHS $\subset$ RHS.

(2) Let $x \in$ RHS. So that $x \in \bigcup_{i=1}^{n} \text{Project}(Ri)$. Then for some j between 1 and n, $x \in$ Project (Rj). Thus $\exists$ $r \in Rj$ such that $x = \text{Project}(r)$. Since $r \in Rj$, then $r \in \bigcup_{i=1}^{n} Ri$, so that

$$x = \text{Project}(r) \in \text{Project}\left(\bigcup_{i=1}^{n} Ri\right) = \text{Project}(R).$$

Therefore RHS $\subset$ LHS.

Since the LHS $\subset$ RHS and RHS $\subset$ LHS, they are equivalent.

Min

$$\text{Min}(R) = \text{Min}\left(\bigcup_{i=1}^{n} \text{Min}(Ri)\right)$$

The "min" operation is a "project" operation that returns the value of the smallest tuple within the project. Since the

minimum function is associative and commutative, the grouping and ordering of the tuples is irrelevant.

Max

$$\text{Max}(R) = \text{Max}\left(\underset{i=1}{\overset{n}{\text{Max}}}(Ri)\right)$$

The "max" operation is the same as "min" except that the largest value is returned.

Sum

$$\text{Sum}(R) = \sum_{i=1}^{n} \text{Sum}(Ri).$$

The "sum" operation uses an attribute list to determine which fields to sum. Since addition is associative and commutative, the grouping and order is irrelevant.

Sort

$$\text{Sort}(R) = R.$$

Because the "sort" operation simply reorders the relation without changing the set, the set is equivalent regardless of ordering.

Product

$$\text{Product}(R, S) = \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Product}(Ri, Sj) \right).$$

Proof: Show that (1) LHS $\subset$ RHS, and (2) RHS $\subset$ LHS.

(1) Let $x \in$ LHS. So $x \in$ Product (R, S) thus $\exists\ r \in R$ and $\exists\ s \in S$ such that $x = \text{Product}(r, s)$. Since $R = \bigcup_{i=1}^{n} Ri$ and

$r \in R$, then for some k between 1 and n, $r \in Rk$. Since

$S = \bigcup_{j=1}^{m} Sj$ and $s \in S$, then for some 1 between 1 and m, $s \in S1$.

Then Product $(r, s) \in$ Product $(Rk, S1)$, and

Product $(Rk, S1) \subset \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Product } (Ri, Sj) \right)$. Thus

$x = \text{Product } (r, s) \in \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Product } (Ri, Sj) \right)$.

Therefore the LHS $\subset$ RHS.

(2) Let $x \in$ RHS. So $x \in \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Product } (Ri, Sj) \right)$. Then for

some k between 1 and n, and some 1 between 1 and m,

x Product $(Rk, S1)$. Thus $\exists\ r \in Rk$ and $\exists\ s \in S1$ such that

$x = \text{Product } (r, s)$. Since $x \in$ Product $(Rk, S1)$, then $r \in \bigcup_{i=1}^{n} Ri$

and $s \in \bigcup_{j=1}^{m} Sj$. So $x = \text{Product } (r, s) \in \text{Product } \left(\bigcup_{i=1}^{n} Ri, \bigcup_{j=1}^{m} Sj \right)$

= Product (R, S). Hence RHS $\subset$ LHS.

Since the LHS $\subset$ RHS and RHS $\subset$ LHS, they are equivalent.

Join

$$\text{Join } (R, S) = \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Join } (Ri, Sj) \right).$$

The proof for "join" is the same as for "product".

Intersect

$$\text{Intersect } (R, S) = \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{m} \text{Intersect } (Ri, Sj) \right).$$

Because the "intersect" operation is a special equivalence join, where the order and number of attributes in both relations R and S are identical, it is proven by the "join" operation's proof.

Diff

$$\text{Diff } (R, S) = \bigcup_{i=1}^{n} \text{Diff } (Ri, S).$$

Proof: Show that (1) LHS $\in$ RHS and (2) RHS $\subset$ LHS.

(1) Let $x \in$ LHS. So that $x \in$ Diff (R, S), thus $\exists$ $r \in R$ such that $x = \text{Diff } (r, S)$. Since $R = \bigcup_{i=1}^{n} Ri$, and $r \in R$, then for some j between 1 and n, $r \in Rj$. Thus Diff $(r, S) \in$ Diff (Rj, S). Since Diff $(Rj, S) \subset \bigcup_{i=1}^{n}$ Diff (Ri, S), then

$x = \text{Diff } (r, S) \in \bigcup_{i=1}^{n} \text{Diff } (Ri, S)$. Therefore the LHS $\subset$ RHS.

(2) Let $x \in$ RHS. So, $x \in \bigcup_{i=1}^{n} \text{Diff } (Ri, S)$. Then for some j between 1 and n, $x \in$ Diff (Rj, S). Since $x \in$ Diff (Rj, S), then $\exists$ $r \in Rj$ such that $x = \text{Diff } (r, S)$. Since $r \in Rj$, then

$r \in \bigcup_{i=1}^{n} Ri$, so that $x = \text{Diff } (r, S) \in \text{Diff } (\bigcup_{i=1}^{n} Ri, S)$

$= \text{Diff } (R, S)$. Thus RHS $\subset$ LHS.

Since LHS $\subset$ RHS and RHS $\subset$ LHS, they are equivalent.

Union

The purpose of the "union" operation is to join files together. Because of this, it will not be split across processors.

Insert

The "insert" operation is the same as "union".

V Local Hardware and Restrictions

Introduction

Rather than purchase the expensive specialized hardware necessary to implement a complete backend database system, the initial design will be performed on equipment currently present at AFIT. Once a working model of the backend is available, it will be easier to determine specific hardware requirements needed to improve performance. This chapter will discuss the available hardware at AFIT, and the compromises necessary in the overall design of the system.

Developement Hardware

The initial version of the BCP was developed on an S-100 system with the CP/M version 2.2 operating system, and the BDS 'C' compiler. This should make the BCP transportable to any system capable of running CP/M and with minor modifications to any system supporting a 'C' compiler. The S-100 system included:

* a dual 8 inch double density floppy disk drive
* Advanced Digital Corporation "Super Quad" card (1) with:
  - Z-80A cpu (4 MHZ)
  - Floppy disk controller
  - 64K of dynamic memory
  - 2 serial ports
  - 2 12 bit parallel ports

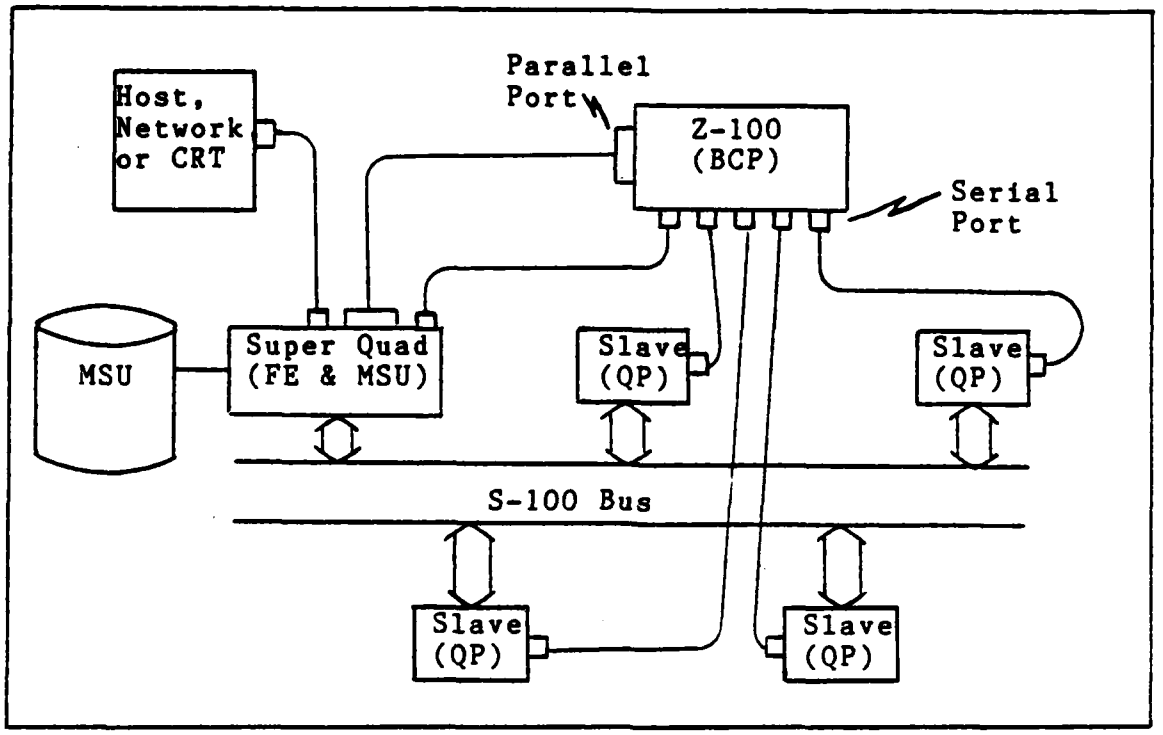


Figure 3. Initial Developement System Configuration.

Initial Configuration

The BCP was designed so that upon completion, it could be moved to one of the Z-100 systems. The S-100 system could be upgraded by the addition of four Advanced Digital Corporation "Super Slave" cards to create a multi-processor system. The remaining components of the backend would be housed within the S-100 system. The "Super Quad" card would become the FE and the MSU, and each "Super Slave" card would become a QP. The floppy disk would provide the physical storage (See Figure 3). Each QP will communicate with the BCP through a serial port, and send/recieve data pages over the S-100 bus. The BCP is also connected to the "Super Quad" (FE) card via a

serial and parallel port. The serial port will be used to send disk commands, while the parallel port will be used for communications with the FE. The reason for using separate ports is to create a logical difference between the FE and the MSU to allow for future expansion.

Drawback

There are two major drawbacks to the initial system configuration. First, all disk I/O must pass thru the FE processor. Second, there is no MBU.

Initially, the FE and the MSU (disk) will both be on the "Super Quad" card. During initial development (i.e. while an actual database does not reside on disk), this should not present any problem. But once the MSU actually begins paging data in and out of the QPs, this processor will become the bottleneck of the backend system. Even without the FE on the "Super Quad" card, it is expected that the MSU will be unable to supply the data pages to the QPs as fast as the QPs can process them. This means that even without the extra burden the FE places on the "Super Quad" card, the QPs will frequently be idle waiting for data pages.

The second problem with the initial configuration is the lack of an MBU. Because of this, a minimum of eight logical pages of memory in each QP must be reserved for data buffer space. This restriction interferes with the backend's ability to allow pipeline query step processing. The "Super Slave" cards are not able to activate the S-100 bus, so data

may not freely pass between QPs. This means that as a QP finishes a page of output data, the MSU must read the page, and rewrite it into the follow-on QP data buffer. Because the paging algorithm was designed to operate on shared pages in the MBU, and not copy data between buffer pages, the initial version of the BCP will not handle the pipelining of query steps. This restriction should be corrected at the earliest opportunity since efficient pipelining would ease some of the MSU bottleneck.

Alternative Configuration

An alternative configuration is included here because of the notable shortcomings of the original configuration. This alternative configuration is provided more as a thought provoking concept than as a solution. It is hoped that any follow-on investigators will accept the better ideas presented here, and reject those for which they can devise a superior architecture.

The alternative configuration includes an additional micro-processor with its own disk drive (See Figure 4). The FE would be placed on the new processor, and the database data dictionary would be stored on the disk. This would allow the FE quick access to any information needed for query optimization while reducing the burden of the database MSU. The "Super Quad" card would then contain the MSU driver, while the bulk of memory on the "Super Quad" card would then

become the MBU.

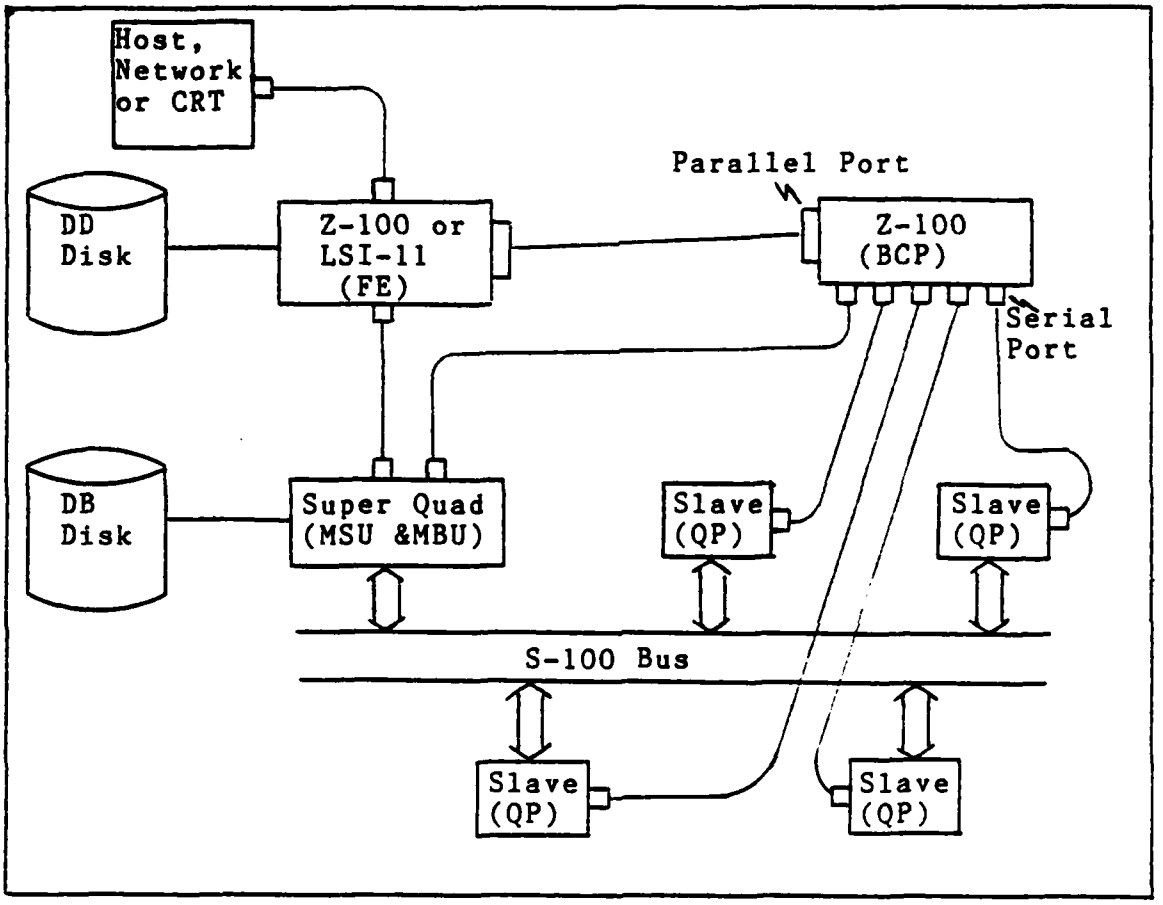


Figure 4. Alternative Developement Configuration.

VI Data Structures

Introduction

There are numerious data structures used throughout the BCP. For the reader to gain a firm understanding of the BCP software, one must first understand the data structures and the utilities provided to manipulate them. This chapter will discuss the following major data structures plus the sub-structures of which they are composed

* task tree
* message queues
* frontend structures
* query processor structures
* mass store unit structures
* system status structure

Each structure will be discussed in a bottom-up approach, followed by a short explanation of modules designed to operate on the structure.

Task Trees

There are two types of task trees used within the BCP. The first one is called "task_tree" which contains all active queries (with their query steps & files). The second is the "stopped_job" which has all the queries that have been temporarily halted.

The task trees are built up from query branches, which contain query trees. Each node in the query tree (called query nodes) contains a query step and has pointers to its input/output relations (See Figure 5).

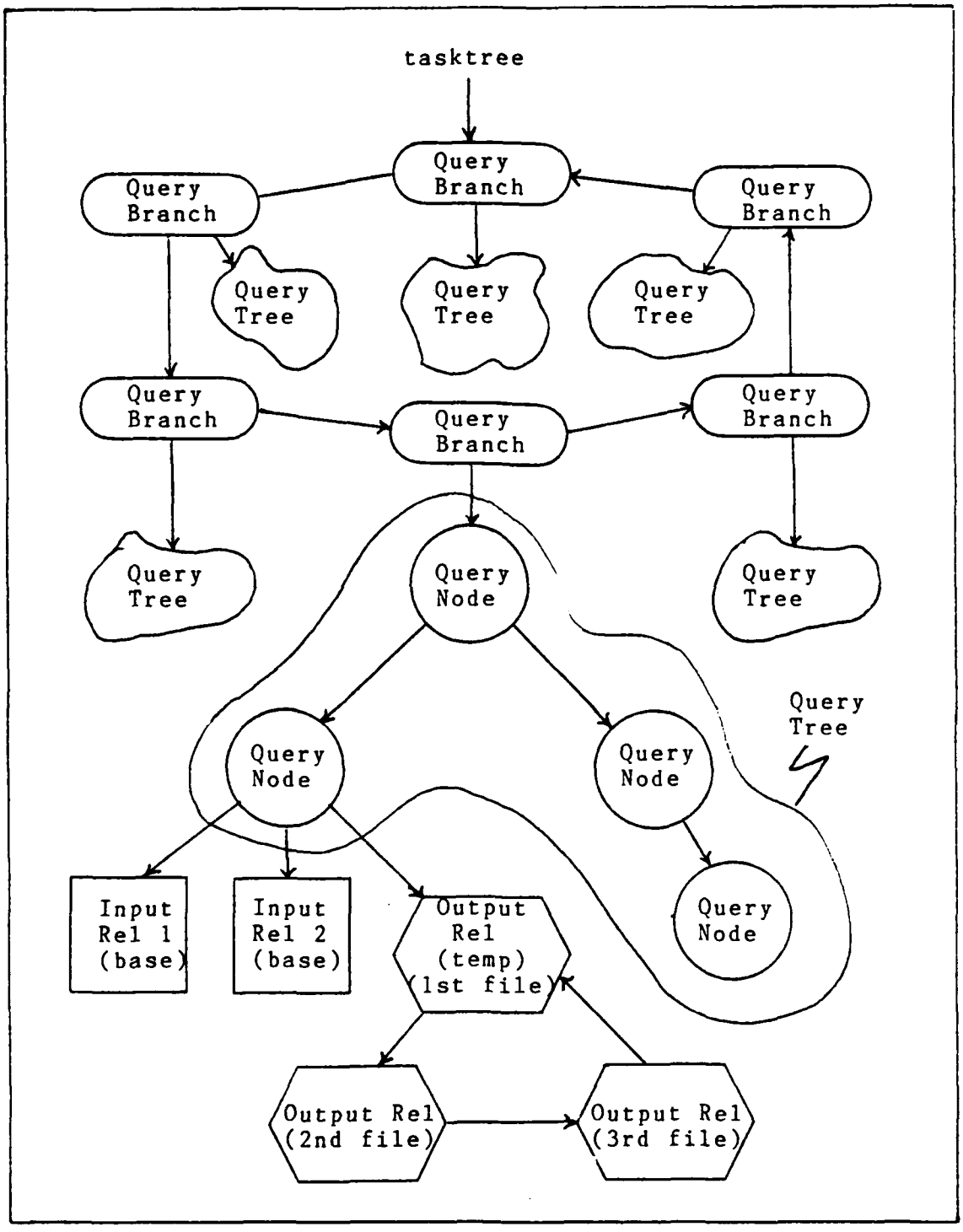


Figure 5. Overview of Tasktree Structure.

Within the BCP, there are two types of relations. Base (permanent) relations, and temporary relations. Each of these are treated differently at the lowest level.

Relational Database Files (base relations)

The base relations have all their pertinent data stored in the "base_rel" structure. This structure contains:

rel_name -- name of the base relation.

sorted -- boolean flag:
TRUE ==> relation is sorted,
FALSE ==> relation is not sorted.

rel_size -- number of logical pages in the relation.

File List (temporary relations)

A file list is a logical relation. When several QPs operate on the same query step, each QP qenerates its own output file. Rather than physically join the separate files (requiring additional I/O paging), the files are logically combined through a file list.

The "file_list" is a circular doubly linked list of temporary output relations. It consists of:

prevfile -- pointer to previous file in linked list.

nextfile -- pointer to the following file in the linked list.

filesize -- size of file in logical pages.

file_id -- file identifier used by 'C' open function.

status -- QP id of QP writing to/reading from the file (-1 if none).

filename -- name of the file being written/read.

Query Node

The "query_node" structure takes all the data related to a single query step and consolidates it in one location. Thus there is a one-to-one correspondence between query steps and query nodes. Because of this, the terms "query step" and "node" are often interchanged within the code documentation.

All the query nodes of a single query are stored in a modified binary tree structure (See Figure 6). The query node is used to connect all the query steps of a query in a logical form, and maintain information about the input/output relations of each query step. The query node consists of:

parent -- pointer to parent node (NULL if root node).

rchild -- pointer to right child (NULL if no right child).

lchild -- pointer to left child (NULL if no left child).

prevleaf -- pointer to previous bottom-most leaf (NULL if this node has any children, or is the first bottom leaf).

nextleaf -- pointer to next bottom-most leaf (NULL if this node has any children, or is the last bottom leaf in the query).

branchp -- pointer to query_branch of this query.

stepp -- pointer to the query step for this node.

relation [3] -- pointer to the three relations (files) accessed by this query (two input, one output).

file_type [3] -- indicates if a relation is permanent or temporary.

rel_sorted [3] -- indicates if the relation is currently sorted.

error_file -- pointer to the error file.

number_qps -- number of QPs currently working this query step.

node [0] -- dummy variable - this is where the query step is actually stored.

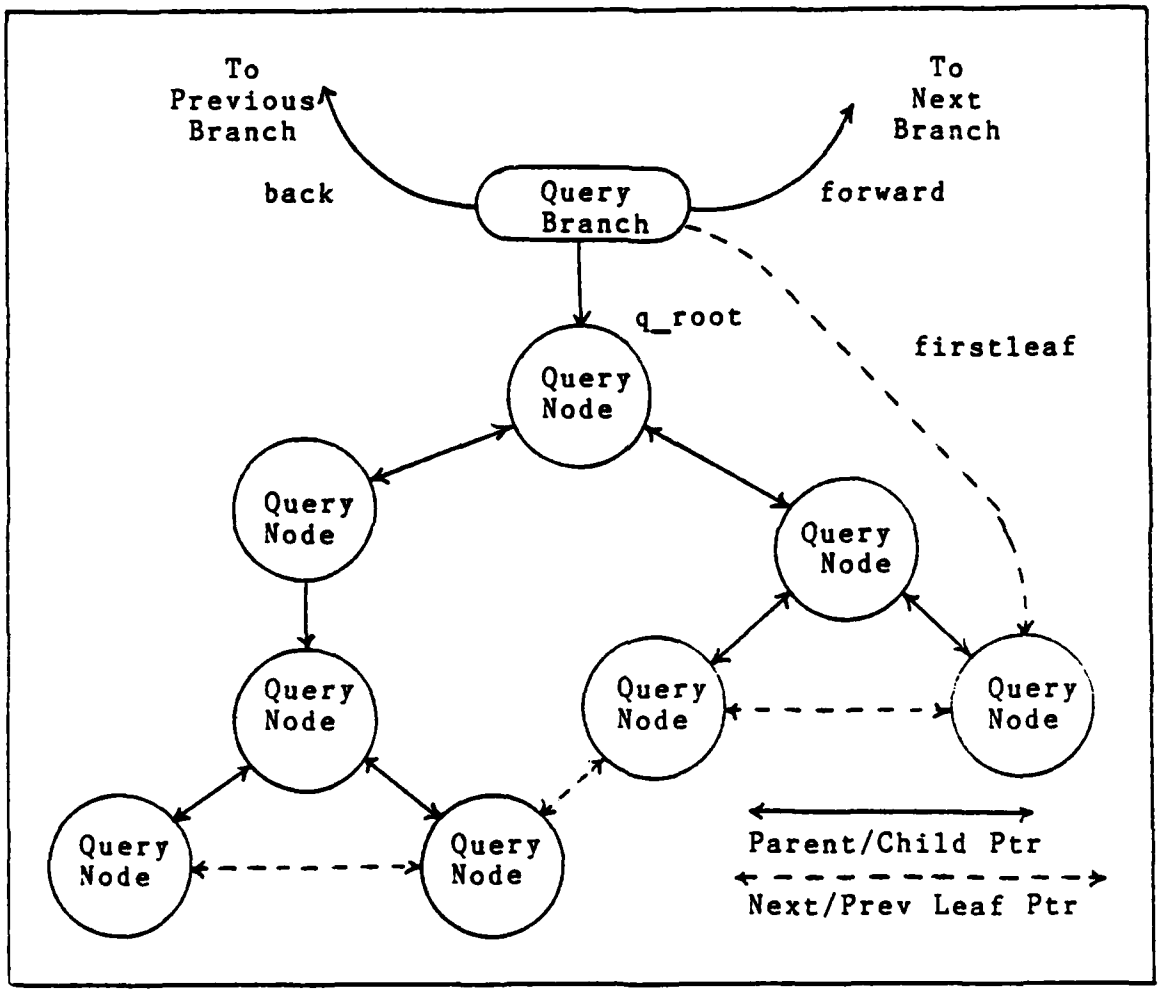


Figure 6. Query Tree (Branch).

Query Branch

The query branch takes all the data related to a query and consolidates it in one place (See Figure 6). There is a

one-to-one correspondence between the query and the query branch. It contains:

back -- pointer to previous branch.

forward -- pointer to next branch.

q_root -- pointer to root node of query step (query node) tree.

first_leaf -- pointer to first leaf without children.

head_node -- pointer to query header.

bot_count -- number of bottom-most leaves in the query.

tag_id -- tag number of next file for this query. As files are needed for temporary storage, they are created by concatenating <job_id> "." <tag>, where <job_id> is the query job identifier, and <tag> is the tag identifier. "tag_id" ranges from 0 to 999. If this should prove to be insuffient, alpha-numeric tags could be used instead of numberic tags.

last [0] -- dummy variable - this is where the query header information is actually stored.

Task Tree

The task tree contains all active queries with their associated query step and files (relations). It is simply a pointer to the highest priority query branch (See Figure 7). The overall structure of a Task Tree is as follows:

* the task_tree points to the highest priority query branch in a circular doubly linked list
* each branch has a query header and query tree
* each node in a query tree has a query step and several file lists
* each file list contains the names of all the physical files that make up the logical file

Since the task tree structure is so complex, it requires special modules to simplify its use. Below is a brief discription of existing modules designed to act on the task tree

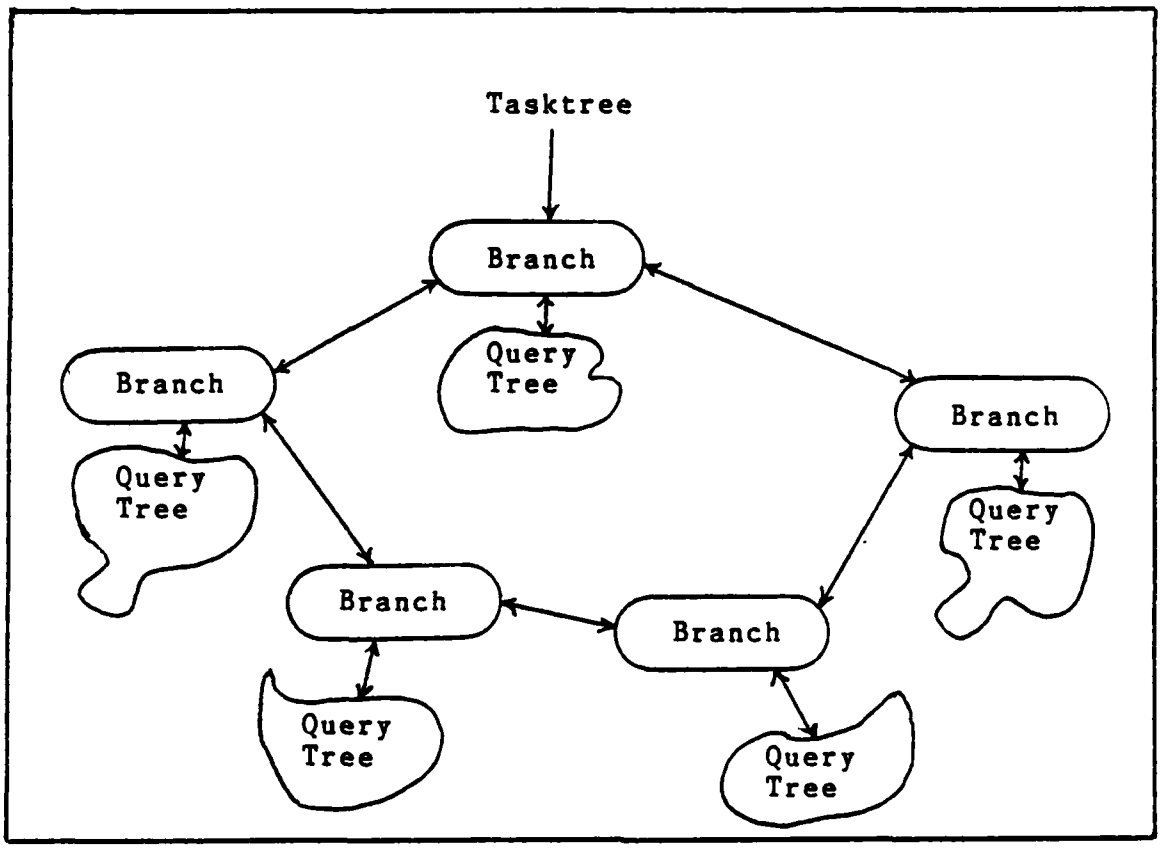


Figure 7. Highlevel View of Tasktree.

and its components.

rebuild_query_tree (FE_msg, Branch)
Takes an input query from the FE and rebuilds it into a tree form (query branch), returns pointer to the root node.

place_in_task_tree (task_tree, branch)
Adds a query branch to the task_tree. Returns task_tree.

disconnect_branch (task_tree, branch)
Removes a query branch from the task_tree. Returns task_tree.

trim_branch (task_tree, branch)
Deletes a query branch from the task_tree, and frees up any storage used by the branch and its

components (query_nodes, file list, etc.).
Returns task_tree.

trim (task_tree)
Deletes the entire task_tree and frees all storage of its components.

count_leaves (task_tree)
Returns the number of bottom-most leaves in the entire task_tree.

select_high_leaf (task_tree, leaf)
Returns pointer to the next highest priority leaf after the input leaf (if input leaf is NULL, return pointer to the highest leaf in task_tree).

kill_subtree (query_node, rel_index)
Deletes and frees all nodes in the sub-tree of the input node. Deletes and frees temporary files with an index less than rel_index. By setting rel_index equal to two, only input relations are deleted, leaving the output relation intact so the next query step can use it for input. By setting rel_index to three, all files, input and output are freed.

trim_leaf (leaf)
Deletes and frees a bottom-most leaf and its input files. Relinks the output file to the parent node as an input relation. Return boolean; TRUE if query is complete, FALSE otherwise.

free_files (file)
Frees storage of a logical file, deletes temporary relations from disk.

remove_file (file, leaf)
Deletes a single physical file from a logical file. Returns pointer to first physical file of the logical file.

npages (leaf, rel_id)
Returns number of logical pages in file indicated by the rel_id.

get_file_name (branch, name)
Sets name to a unique file name.

Also included for debug purposes are:

dump_tt (task_tree)
Dump the entire task_tree in hexadecimal.

dump_branch (query_branch)
Dumps the query branch.

dump_subtree (node)
Dumps the sub-tree beginning at input node.

dump_rel (node)
Dumps the relations of the input node.

dump_files (relation)
Dumps the file list of a temporary relation.

Message Queue

The message queue is a single linked list priority queue for holding messages passed between the BCP and the other system components. There are two message queues in the BCP; one for incoming messages, and one for outgoing messages. The structure of the message queue consists of:

m_type -- component type (FE, MSU, or QP) of source or destination.

m_id -- component identifier number (because of multiple QPs).

m_time -- (test value) - time message was sent/received. This is the current priority field.

m_ptr -- pointer to the message.

m_next -- pointer to next message structure in the queue.

The following modules act on the message queues.

add_queue (queue, message)
Allocate storage for message and place the message into the queue. Returns pointer to top element of queue.

remove_queue (queue)
Removes top element from queue and frees storage. Returns pointer to top element.

read_queue (queue)
Returns pointer to top element.

FE Structures

The FE must pass two types of messages to the BCP; command and query messages. Because of time limitations, the BCP command handling routines have not been designed in enough detail to create a command structure. Thus no structure is shown for the command message. The query message (See Figure 8) consists of a "query_header", following by an array of "base_rels" (one per base relation accessed), and an array of "query_steps" (one per step query in the query). The "base_rel" structure has already been discussed in detail under the "task_tree" structure.

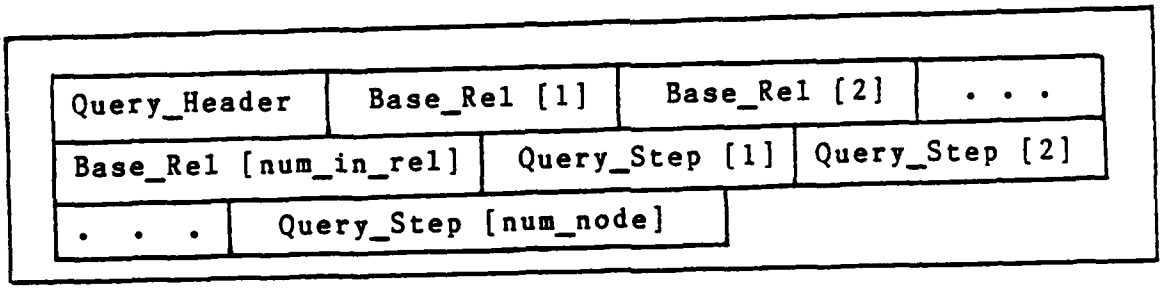


Figure 8. FE Query Message.

Query Step

A query step is a single operation to be performed on a relation(s). The operations are discussed in Chapter 3. The "query_step" structure contains all the information needed by the QP to perform the specific query step operation on the assigned pages in the MBU.

Because this is a complex structure and subject to evolve, a version number is included to allow the backend system to work with multiple versions of "query_step" structures if needed. The BCP only uses the header portion of the "query_step" structure, the remaining sections may be freely modified by the FE and/or QP design with no effect to the BCP's performance.

Query Step -- Header

qs_version -- version number of the query step (should be zero)

len_step -- number of bytes needed to store the query step. Because the query step structure varies in length depending upon the selection criterion, modification list, and attribute list, this field is included to enable the BCP and the QPs to know the exact size of each query step.

type -- type of query step operation (i.e. "Select", "Project", "Join", etc.).

rel_type [3] -- three rel_types are included because a query step may have two input relations and one output relation. The "rel_type" field is used to distingish between base (permanent) and temporary relations. For "unary" relational operation, the second input file is set to NULL.

rel_ptr [3] -- this is really an index node number, because it is impossible to pass the actual pointer value between two processors without shared memory. When the FE passes the Query Tree to the BCP, it numbers the query steps

in the order that they are sent. Later, as the BCP rebuilds the Query Tree, the actual pointer values are computed.

Note: rel_type and rel_ptr are used by the BCP to rebuild the Query Tree. The QPs do not use these fields.

dups_ok -- a boolean flag to determine if duplicates are allowed in the query step.
TRUE ==> keep duplicates,
FALSE ==> remove all duplicates.

ratio [2] -- (for testing BCP only) this field is used by the "fake_qp" module to simulate the reduction of data for a query step. It contains two percentage numbers: minimum and maximum amount of data. Two examples of its use are given here:

Example 1: On a "sort" operation, nearly all data in a page is retained (duplicates are removed), so by setting ratio [0] = 99, and ratio [1] = 100, the fake QP would retain between 99 and 100 percent of all input data in the output relation.

Example 2: On a "select" operation, a large percent of the data will be removed. If the ratio were set to 0 and 100 respectively, then each page of the input relation might be completely empty or full of "useful" data. The average would be 50 percent of the data being retained.

Query Step -- Selection Criterion

The remainder of the "query_step" structure is not used by the BCP and will be dictated by the QP's needs and what the FE can supply. What is given below is the author's view of the necessary information .

num_ands -- the number of AND conditions which must be met to evaluate the selection criterion as TRUE.

len_and [num_ands] -- the length of each individual AND condition. This is provided so that the remaining part of an AND condition can be skipped if a single OR condition inside the AND evaluates to TRUE.

num_ors [num_ands] -- the number of OR conditions within each AND condition. Note: an AND condition without an OR is treated as if it had one OR condition.

Example Selection Criterion for a "select" operation:

```
select *
where salary < 8000 & (job = "clerk" ½ job = "student" ½
   job != job_title) & (state < "LA" ½ country != "USA")
```

This selection criterion has three AND conditions. The first AND has no OR condition, but is treated as if it has a single OR condition. The second AND has three ORs, and the last AND has two ORs. In any AND condition, as soon as an OR condition evaluates TRUE, the remaining portion of that AND may be skipped. Likewise, if any AND evaluates FALSE, the selection criterion evaluates to FALSE, and further checks may be skipped.

For each OR condition inside of an AND condition, an attribute must be compared against some other value. The rest of the selection criterion fields deal with determining which attribute to compare to what value.

field_type [num_ands, num_ors] -- type of data field. It must be one of the following:

- c -- character string
- d -- double precision
- f -- float
- l -- long integer
- i -- integer

Note: BDS 'C' only supports character and integer variables. Therefore, the initial version of the backend system will only support these two data types.

len_field [num_ands, num_ors] -- gives the length of an attribute field in bytes for fixed sized tuples. Otherwise it is set to zero for relations with varying length tuples.

loc_field [num_ands, num_ors] -- gives the starting byte number in fixed relations. Gives the field number in varying relations.

Example of fixed relation:

If relation 'X' were defined as:

```
supply    int,
job       char (10),
job_title char (10),
state     char (2),
country   char (10);
```

Then if attribute field "job_title" were being used in a selection criterion, then

```
field_type = 'c',
len_field = 10,
loc_field = 12;
```

Example of varying relation:

If all (or any one) of the character strings in relation 'X' were declared as varying, then 'X' is considered a varying length relation, and the attribute field "job_title" would be:

```
field_type = 'c',
len_field = 0,
loc_field = 3;
```

Note: the len_field and loc_field fields are collectively known as the attribute identifier.

oper [num_ands, num_ors] -- comparison operation (i.e. "=", ">", "<", "!=", ">=", and "<=").

field_or_constant [num_ands, num_ors] -- determines if the comparison is against another attribute field or a supplied constant.

"f" ==> field comparison;
"c" ==> constant comparison.

Note: for "join" operations, this should always be a field to field comparison, and the first field should be from the first input relation, while the second field should be from the second input relation.

len_f_or_c [num_ands, num_ors] -- if field_or_constant = = "f" then this is the same as len_field. Otherwise, this is the length (in bytes) of the following constant value.

f_or_c [num_ands, num_ors] -- if field_or_constant = "f", then this is the same as loc_field. Otherwise, this is the constant value being compared.

Query Step -- Modification List

The modification list given here limits the user to assigning fixed values to attribute fields. The limit should be removed so that a computed value may also be used if desired.

num_mods -- number of fields to modify.

field_type [num_mods] -- same as field_type in selection criterion.

len_field [num_mods] -- same as len_field in selection criterion.

loc_field [num_mods] -- same as loc_field in selection criterion.

len_new_value [num_mods] -- length (in bytes) of the new value to be placed in the attribute field.

new_value [num_mods] -- the new value to be placed in the attribute field.

Query Step -- Attribute List

num_atrib -- number of attribute fields to retain in a project like operation.

field_type [num_atrib] -- same as field_type in selection criterion

len_field [num_atrib] -- same as len_field in selection criterion.

loc_field [num_atrib] -- same as loc_field in selection criterion.

Query Header

When a query is passed from the FE to the BCP, it has a query header containing the information needed by the BCP to rebuild the query into a query tree, and to access the necessary base relations.

h_version -- current header version number (should be zero).

num_node -- number of query steps (nodes) in this query.

len_head -- length of query header (including "base_rel" array) in bytes.

len_msg -- length of the entire query message including all the query steps.

num_in_rel -- number of input (base) relations accessed.

priority -- priority of this query (0 - highest, 255 - lowest).

job_id -- job identifier (must be unique).

rel_info [0] -- dummy variable - this is where the "base_rel" array is placed. There is one "base_rel" structure for each base relation accessed.

The FE query message is build into a "query_branch" structure by calling:

rebuild_query_tree (FE_msg, branch)
Takes an input query message from the FE, and rebuilds it into a tree form (query_branch). Returns a pointer to the root node.

QP Structures

There are four structures in the BCP that deal with the QPs. They are:

* query_step -- query step for the QP to execute
* qp_page_info -- paging information
* qp_status -- status of each qp
* query_processor -- (test only) fake QP simimulation

The "query_step" structure has been discussed under FE structures. This structure is passed to the QPs to direct the type of relational operation to perform. It includes the select criterion, the modification list, and the attribute list.

QP Paging Information

The "qp_page_info" is used between the QPs and BCP to pass paging information. The BCP uses it to direct the QPs

as to which pages in the MBU to act against, and what type of data is in each page (input rel 1, input rel 2, or output rel). The QPs use the structure to request new input/output pages. It consists of:

qp_version -- current message version (should be zero).

len_qp_msg -- length of paging message.

qp_msg_type -- paging info flag. This byte distinguishes a paging structure from a query step structure for the QP.

qp_id -- QP identification number.

buff_addr -- page in the MBU to access.

page_type -- type of data within the page (rel 1, rel 2, or output rel).

eof -- EOF flag. The BCP sets this if this is the last page of the query step. The QP returns this flag after completing the final input page.

results -- results of a value returning operation.

QP Status

The "qp_status" structure is used by the BCP to keep track of which QPs are working on what query steps, and which page to send next.

active_qp -- actual number of QPs up and running.

free_qp -- number of idle QPs.

qp_idle [MAX_QP] -- boolean flag for each QP to determine if it is currently idle or busy.

buff_allocation [MAX_QP] -- boolean flag to determine if the QP has been allocated pages in the MBU.

qp_step [MAX_QP] -- pointer to current query step being processed by the QP.

qp_start_page [MAX_QP] [2] -- starting page to process within each input relation.

qp_end_page [MAX_QP] [2] -- last page for this QP to process within each input relation.

qp_curr_page [MAX_QP] [2] -- the current page in the QP's buffer space.

qp_log_page_size [MAX_QP] [2] -- a logical page size of each input relation.

qp_file [MAX_QP] -- pointer to output file.

Query Processor

This structure is used to simulate a QP's processing of a query step. It is not used in the actual system.

qs_ptr -- pointer to the query step being processed.

qp_time -- fake clock time.

free_page -- this is an index into qppage, qptype, and percent. It is the next available page location.

eofpage -- flag to determine if this is the last page for this query.

qppage [BUF_RATIO] -- MBU page address.

qptype [BUF_RATIO] -- Type of page (rel 1, rel 2, output rel).

percent [BUF_RATIO] -- percent of the page already processed.

MSU Structures

The MSU receives two types of messages from the BCP; one for paging, and one for commands. These structures are used to direct the paging and file creation/deletion of the MBU.

MSU Paging Information

The "msu_page_info" structure is used to direct the MSU to read/write a page into the MBU. It consists of:

msu_version -- version number (should be zero).

len_msu_msg -- length of the paging message,

msu_msg_type -- this field distinguishes paging information from command messages.

page -- page number in the file to read/write.

msu_id -- disk identification (if needed).

readflag -- flag; TRUE ==> read from file to MBU,
FALSE ==> write to file from MBU.

buf_addr -- page in MBU to read/write.

file -- filename to read/write.

MSU Command Message

The "msu_cmd_msg" structure is used to cause the MSU to create, delete, or concatenate files on the MSU. It consists of:

msucmd_version -- version number (should be a zero).

len_msucmd -- length of command message.

msucmd_msg_type -- flag to distingish command message from paging information.

msu_command -- command to MSU.

n_files -- number of files to act on.

files_name [n_files] -- array of file names.

System Status

The "system_status" structure is used to consolidate all pertinent data of the BCP in one structure. This is the BCP's database to control what is happening within the back-end system. It includes all the structures discussed above plus buffer addresses, and test files.

Buffers

The "buffer" structure maintains status of all input buffers. It contains:

fe_buf -- pointer to FE buffer area.

msu_buf -- pointer to MSU buffer area.

qp_buf [MAX_QP] -- pointer to each QP buffer area.

fe_buf_busy -- flag; TRUE ==> FE buffer is not empty,
FALSE ==> FE buffer is empty.

msu_buf_busy -- flag.

qp_buf [MAX_QP] -- flag for each QP buffer.

Test Files

There are several test files used by the BCP to assist in debugging the program.

query_file -- file containing dummy queries.

fe_in_file -- file which directs which dummy queries to execute.

fe_trace -- trace file for FE messages.

qp_trace -- trace file for QP messages.

msu_trace -- trace file for MSU messages.

rel_index_p -- index to query file page containing queries.

time -- dummy test clock.

System Status (Structure)

shutdown -- flag; TRUE ==> shutdown in progress,
FALSE ==> normal mode.

idle -- flag; TRUE ==> all QPs are idle, and Task Tree is empty,
FALSE ==> has some work to perform.

```
buffer_p  -- pointer to buffer structure.
task_tree -- pointer to Task Tree.
qp_stat   -- pointer to QP status.
query_processor -- pointer to qp_data.
out_que   -- pointer to output queue.
in_que    -- pointer to input queue.
test_files -- pointer to test/trace files.
```

VII Detailed Design

Introduction

This chapter discusses the major algorithms used by the BCP. As stated in Chapter 1, the algorithms are not designed for efficiency, but rather represents a simple-minded approach to make the BCP operational.

When a query step is being assigned to a specific QP, that QP is said to be in a loading phase. During the loading phase, a QP receives a query step message, and several paging messages from the BCP. The QP cannot begin processing the query step until it has received at least one page from each input relation, plus an output page for the results and error file (if needed).

In the initial version, each QP will only hold one query step at a time. Thus there is an idle period between the completion of one query step, and the beginning of the next. To reduce the idle time in the QPs, the assignment algorithm attempts to minimumize the number of query steps it must load for each query. This is done by requiring a QP to complete the entire query step before it can be assigned some other operation. So, once a QP is assigned a query step, it must single-mindedly act on that step regardless of any change of states within the backend system.

This restriction greatly simplifies the intelligence needed by the BCP for QP job assignments and paging, but

reduces the backend system ability to dynamically adjust to the load conditions of the database. It also requires that all input relations of a query step be completed before the query step is assigned to a QP. This eliminates the possibility of pipelining pages through the QPs.

Once the BCP assigns a query step to a specific QP, it must supply input relational pages to the QP, and direct where the output pages are sent. QPs are basically simple black boxes that perform specific relational operations on these pages. This means that if a QP is given a bad input page, it will not realize the error, and will happily perform its assigned task, producing bad output. This being the case, the BCP paging algorithm must be very sophisticated and reliable, or the responses to the user queries will be invalid (thereby making the system useless).

Because the BCP's algorithms are important, this chapter will discuss them in great detail. It begins with the QP assignment algorithm, followed by a discussion on the QP status structure. Next is a rough description of the output control mechanism and the paging system and its algorithms. The final two sections cover buffer allocation, and node splitting.

QP Assignment

The QP assignment algorithm is called any time there is at least one idle QP, and the task tree is not empty. To be

eligible to be assigned to a QP, a query step must be a bottom-most leaf in the Task Tree (See Figure 9).

Step 1) The algorithm first gets a count of the number of eligible query steps, and sets current leaf to NULL (this causes Step 2 to select the highest priority leaf within the Task Tree).

Step 2) Selects the next highest leaf in the Task Tree after the current leaf.

Step 3) If current leaf equals NULL or no more idle QPs, then it exits.

Step 4) If this leaf does not have a QP already operating on it, then it goes to Step 5. Otherwise, if there are other bottom-most leaves (query steps) which have not been assigned to a QP, or this is a "union"/"insert" operation, then it rejects this leaf, and goes to step 2. If all other bottom-most leaves have a QP processing them, then it determines if there are enough pages left to process in this query to warrant an additional QP. If so,it splits the query step across two separate QPs.

Step 5) Assigns the QP to the current leaf, marks the QP as busy, and adds one to the number of QPs acting on this query step.

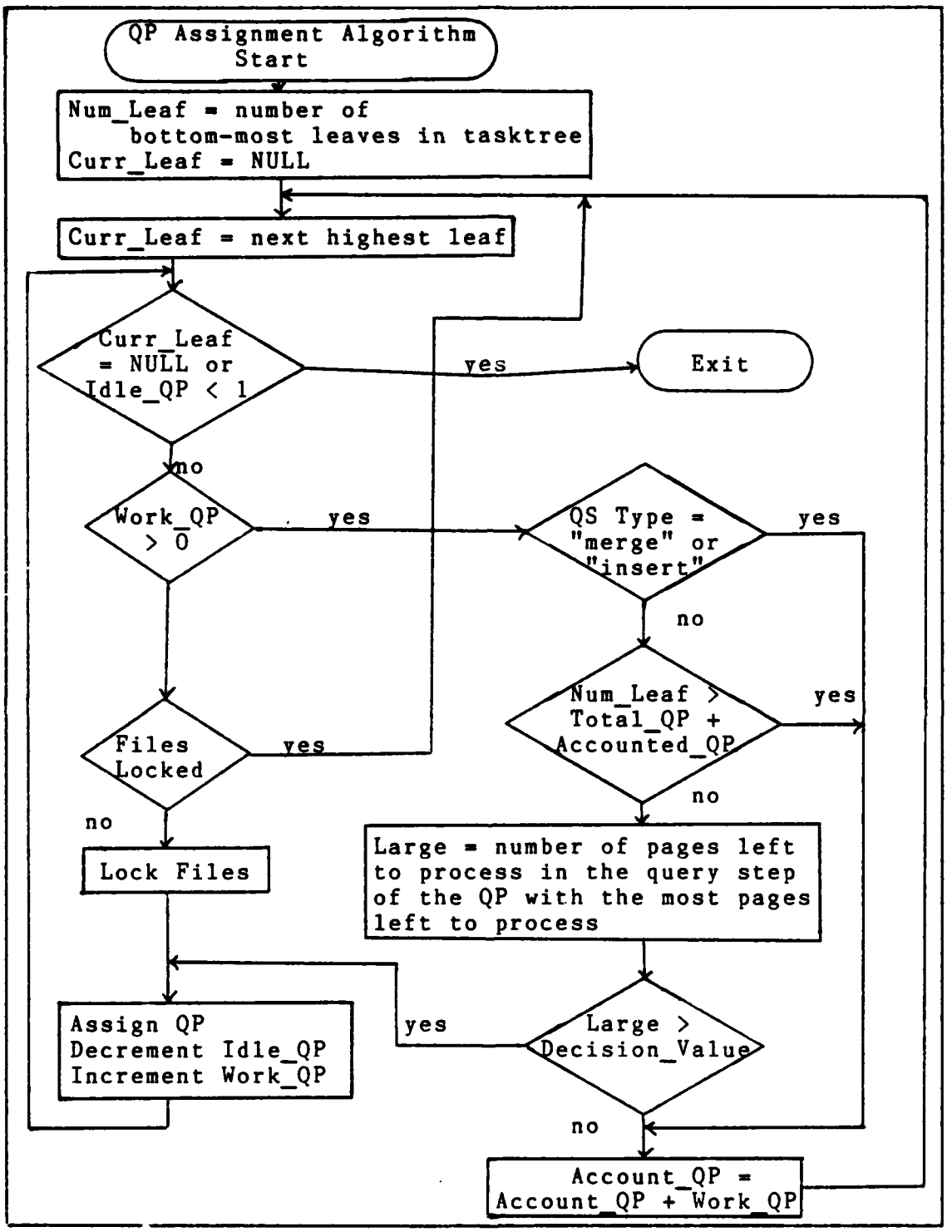


Figure 9. QP Assignment Algorithm

Overview of System Paging

The BCP maintains the following information about each QP to aid in controlling the paging algorithms:

* a pointer to the query step
* a start page index for each input relation
* an end page index for each input relation
* a current page index for each input relation
* a logical page size for each relation
* a pointer to the output file

For the present, the reader only needs a rough idea of how each variable is used. A clearer understanding will be gained after reading over the "Paging Algorithms", and looking at the "Node Splitting" examples.

- The pointer to the query step enables the BCP to quickly determine the query step type, and decide if two QPs are working on the same query step.

- The start page index marks the first page the BCP needs to send to the QP for processing of this query step segment.

- The end page index marks the last page the BCP needs to send to the QP for processing of this query step segment.

- The current page index tells the BCP which page is currently in the QP, and enables the BCP to compute the next logical input page for the QP.

- The logical page size tell the BCP how many pages make up a logical page for each relation.

- The pointer to the output file tells the BCP where to write any output created by the QP.

Output File Control.

Each QP will produce only one response for each query step it is assigned to act on. This means that if only one QP is tasked to process a query step, then the output produced by that QP is the response to the query step. But, if the query step is divided between multiple QPs (See "Node Splitting), then the response is the combined answer of each of the QPs.

For query steps that return a value, each QP will return a single value which the BCP will add or compare to the existing values returned by other QPs working on the query step.

For query steps that produce output relations, each QP assigned to it will generate a unique file. When a QP fills an output buffer in the MBU, it sends a page request to the BCP. The BCP adds the data sequentially to the output file, and provides the QP a new output buffer.

A logical file list is used to logically concatenate the output files of each QP working on a query step. This file list is sorted by ascending page number of the first input relation.

When a query step is completed, it is deleted from the task tree, and any temporary input files are removed from the MSU. The logical output file then becomes an input relation for the parent node. If there is no parent node (i.e. this is the root of the query tree), then the BCP directs the MSU

to concatenate all the files in the logical file into a single physical file. The BCP tells the FE the query has been completed, and gives it the output file name. The FE then passes the file to the host system.

Paging Algorithm

The paging algorithms are used to control the paging of data in and out of the MBU so that the QPs can manipulate the information in the database. The algorithms discussed are simplified to highlight the input paging scheme of the algorithms. This is because the buffer allocation scheme and the output file control are discussed elsewhere in the thesis. The MBU is not discussed because logically it is part of the QP memory area.

In the algorithms, the variable m is the logical page size stored in the QP status structure, and is set by the buffer allocation scheme.

The BCP has three major paging algorithms:

* general paging algorithm -- this is the paging algorithm used by most query step operations
* sort paging algorithm -- this is used only by the "sort" operation
* merge paging algorithm -- this is used by the "union" and "insert" operations

General Paging Algorithm

Step 1) The BCP sets the "start", "end", and "current page" indexes for each input relation. The logical page size is set according to the buffer allocation scheme.

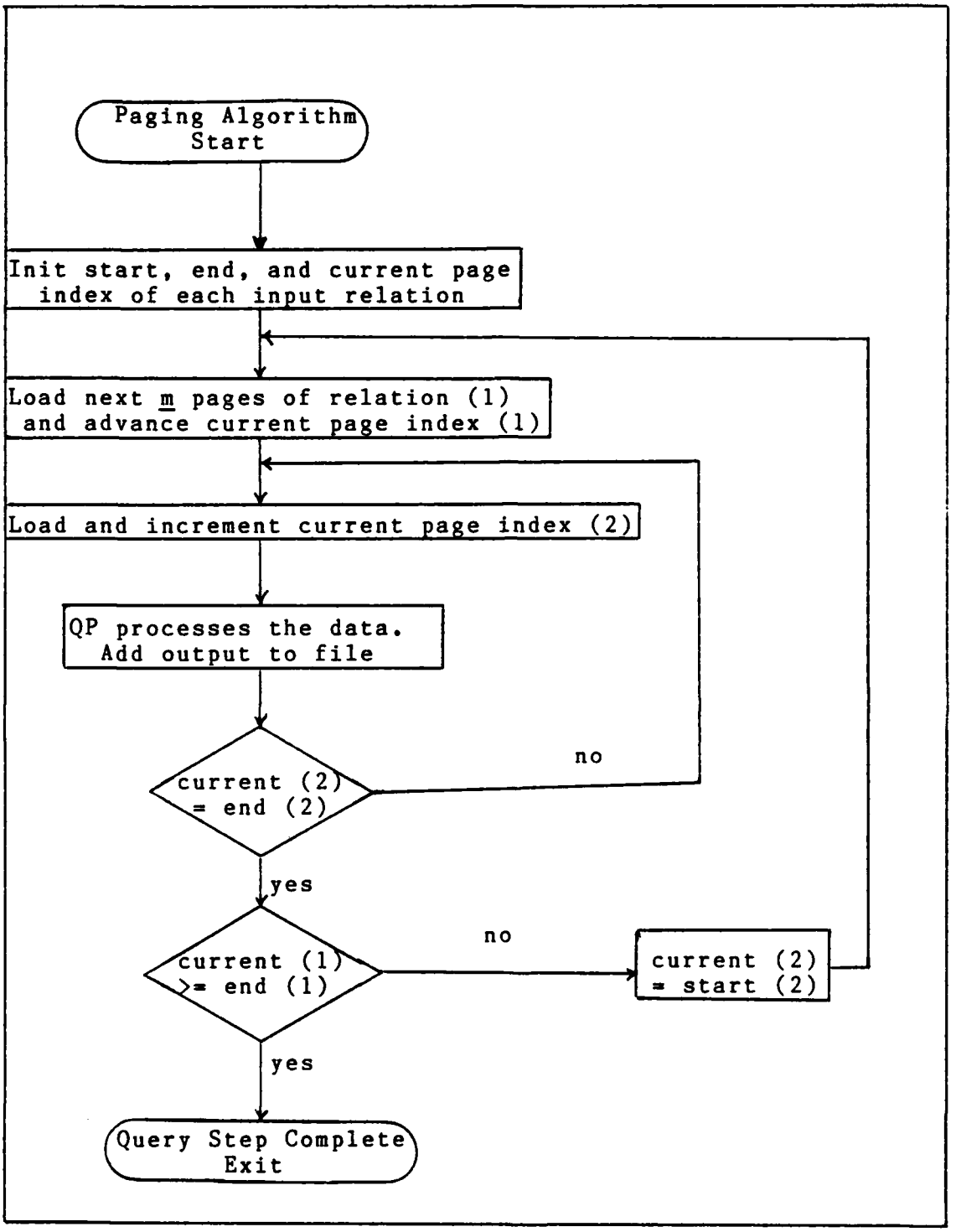


Figure 10. General Paging Algorithm.

Step 2) The BCP directs loading the QP with the next logical page of the first input relation (incrementing the "current page" index as it loads).

Step 3) The BCP then loads the next logical page of the second input relation (if any), and increments the "current page" index.

STEP QP). The QP then operates on the input pages and the BCP stores any output. After the QP has completed its operations on the pages within its memory, it requests additional input pages.

Step 4) The BCP then compares the "current page" index to the "end page" index of the second relation. If they are equal, it means that the m pages of the first relation in memory have been operated on with every page within the "start/end" page range of the second relation. If they are not equal, it goes to Step 3.

Step 5) The BCP compares the "current page" index to the "end page" index of the first relation. If they are equal, the query step is complete. Otherwise, it resets the "current page" index of the second relation to the "start page" index of the second relation and goes to Step 2.

Sort Paging Algorithms

The paging algorithm for the "sort" operation is unlike any other paging algorithm used by the BCP. This is because the "sort" operation requires all the data which is being sorted to be in memory simultaneously, whereas the other

operations are able to work on one page (per input relation) at a time.

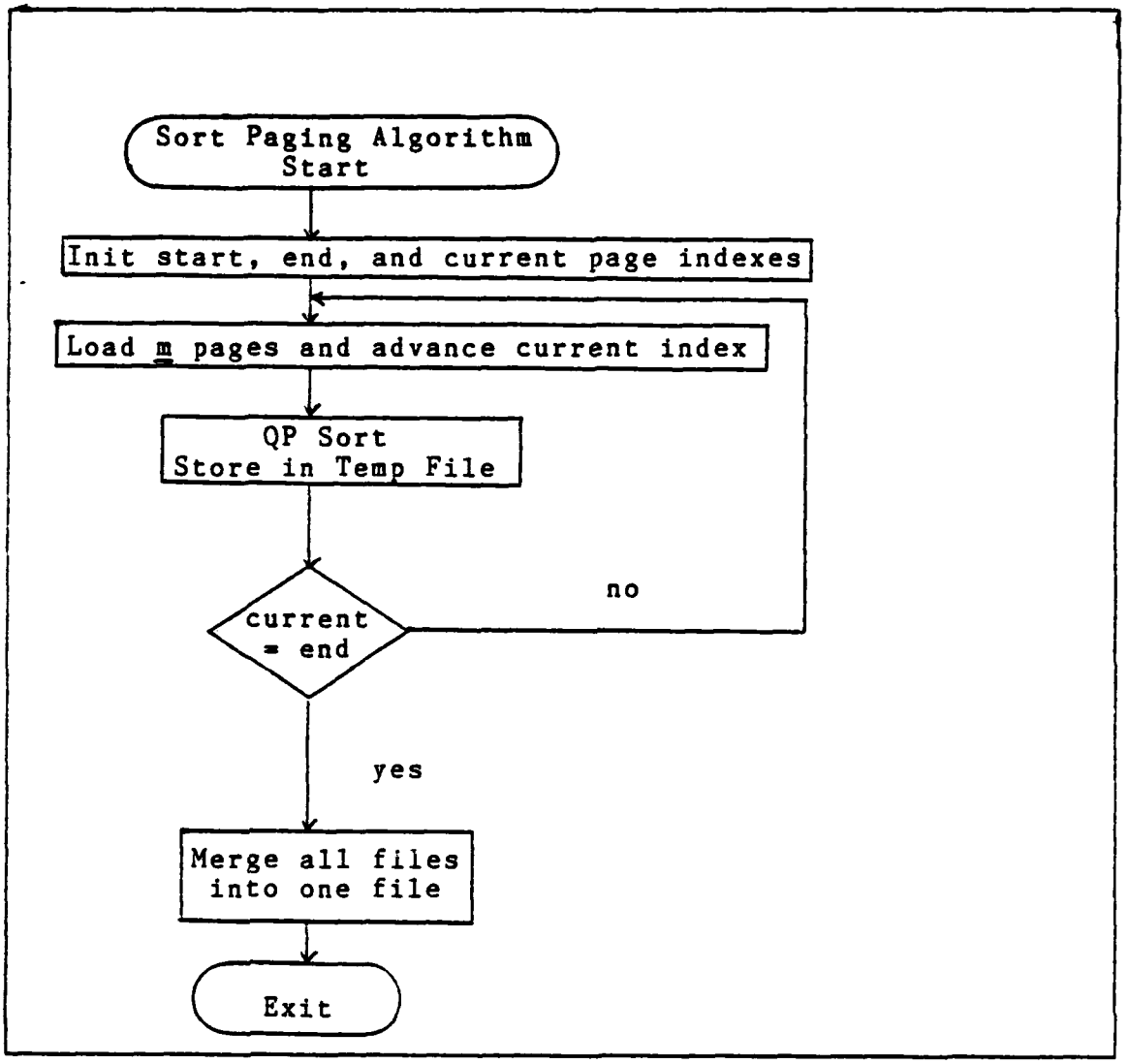


Figure 11. Sort Operation Paging Algorithm.

Step 1) The BCP sets the "start", "end", and "current" page indexes. It sets logical page size to m.

Step 2) It then loads the next logical (m pages) page into the QP and increments the "current page" index accordingly.

STEP QP) The QP does an "in place" sort on the data (i.e. a

Heapsort or QuickSort). These pages are then written into a unique file.

Step 3) The BCP compares the "current page" index with the "end page" index. If they are not equal, it goes to Step 2. This operation continues until the relation consists of many small files.

Step 4) The BCP then uses the "union" operation to merge all the files into a single output relation.

Merge Paging Algorithms

The "union" and "insert" operations use one of two different paging algorithms depending on whether the output relation must be sorted (to eliminate duplicates) or not.

Sorted Merge (eliminate duplicates)

Step 1) The BCP sets the "start", "end", and "current page" indexes of each relation. It sets the logical page size to m.

Step 2) Determines if the two input files are sorted. If an input file is not sorted, create a "sort" node operation to sort the file.

Step 3) The BCP then loads the QP with the next logical page of each relation and increments the "current page" indexes.

STEP QP) The QP uses a merge sort algorithm to combine the two separate files. As the QP completes a page from a relation, it requests another input page from that relation.

Step 4) The next logical page is loaded and the "current

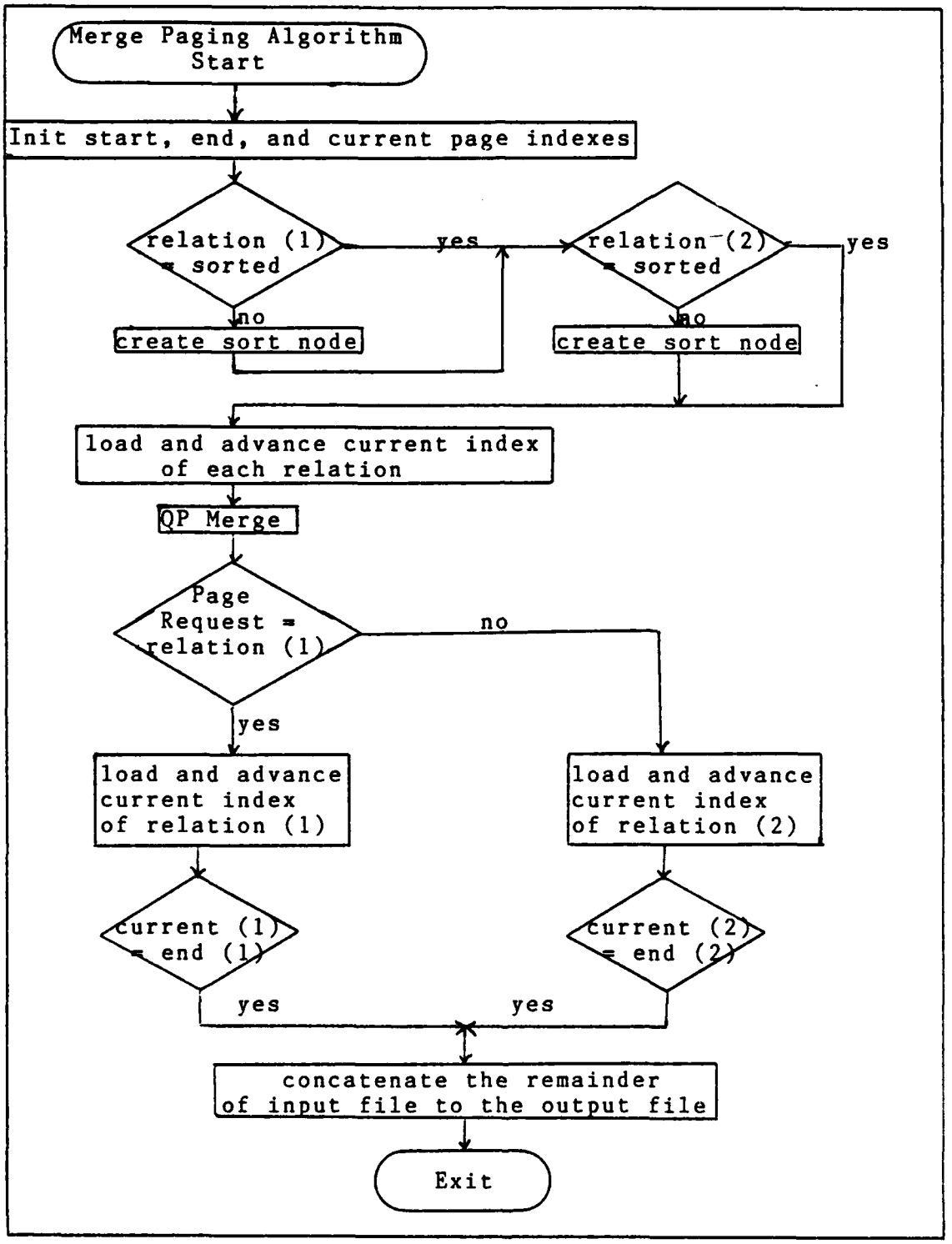


Figure 12. Union/Insert Operations Paging Algorithm.

page" index is incremented for that relation.

Step 5) The "current page" index is compared with the "end page" index. If they are not equal, go to QP STEP.

Step 6) The remains of the other relations are simply concatenated to the output relation.

Unsorted Merge

If the "union" or "insert" operation does not require the data to be sorted, then the two files are simply concatenated together.

Buffer Allocation Scheme

Because the initial configuration does not allow shared pages in the MBU, a deterministic buffer allocation scheme is used rather than a dynamic one. In the initial system, each QP has eight (8) pages of buffer space. The reason eight pages was chosen is because the maximum number of files any one query step operation accesses at one time is four (two input relations, one output relation, and an error file). To reduce idle time waiting for data, double buffering is used in the general case. This results in eight pages of buffer space per QP.

Despite the fact that the initial backend system will use a deterministic buffer allocation scheme, the paging algorithm was designed for a dynamic scheme (in the hope of future enhancements). Because of this, a buffer allocation scheme is needed to assign logical page sizes for the paging

algorithm.

For unary query step operations (except "sort"), nothing is gained by making the logical page greater than the physical page. Therefore, all unary query step operations (except "sort") will have the logical page size set to one.

For the "sort" operation, there is no error file or second relation. Furthermore, the sort algorithm can only sort data currently in memory. Thus it is advantageous to bring in as much data as possible at one time. By using an in-place sorting algorithm (i.e. Heapsort or Quicksort), the entire eight pages can be used for input and output. Hence the logical page size for the "sort" operation is set to eight. Note that this causes an idle time in the QP while swapping in new data (i.e. no buffering), but reduces the paging requirements by the number of pages within the relation being sorted.

For binary query steps (except "union" and "insert"), the pages of the first relation are not swapped until the entire second relation is read. Because of this, very little is gained by double buffering the first relation. Instead, by doubling the logical page of the first relation, the number of times the second relation must be read is halved. Since double buffering is useful for the second and output relations, and there is no error file, four pages are available for the first input relation. Therefore, the logical page sizes for the relations are four and one respectively.

For the "union" and "insert" operation, the next input page could be for either relation. Therefore, double buffering is advantageous, and nothing is gained by changing the logical page size. Both will be set to one.

Node Splitting

When a QP is assigned to a query step, the BCP determines if node splitting is required. If this is the first QP assigned to this query node, then node splitting is not performed. The starting page(s) is zero, and the ending page(s) is the number of pages in the input relation(s). Otherwise, if the BCP is assigning an additional QP ("new QP") to the same query step, then the BCP must split the node into different query step segments. Each segment of the query step is then processed on separated QPs, and subject to further splitting (segmentation).

During node splitting, the output files (if any) are linked in ascending order by their starting page number. The purpose for this is to attempt to maintain sorted relations in a sorted order. This procedure works for query step operations: "select", "delete", "modify", and "diff". It will also work for the other binary operations if the entire first relation is less than m pages.

Note that "insert" and "union" operations may not be split across QPs.

Node Splitting Algorithm

The node splitting algorithm consists of two major steps. The first is to find which QP contains the largest segment of the query step being split. The second step is to actually split the query step segment between the two QPs. The algorithms discussed here are simplified versions of the one used in the BCP software.

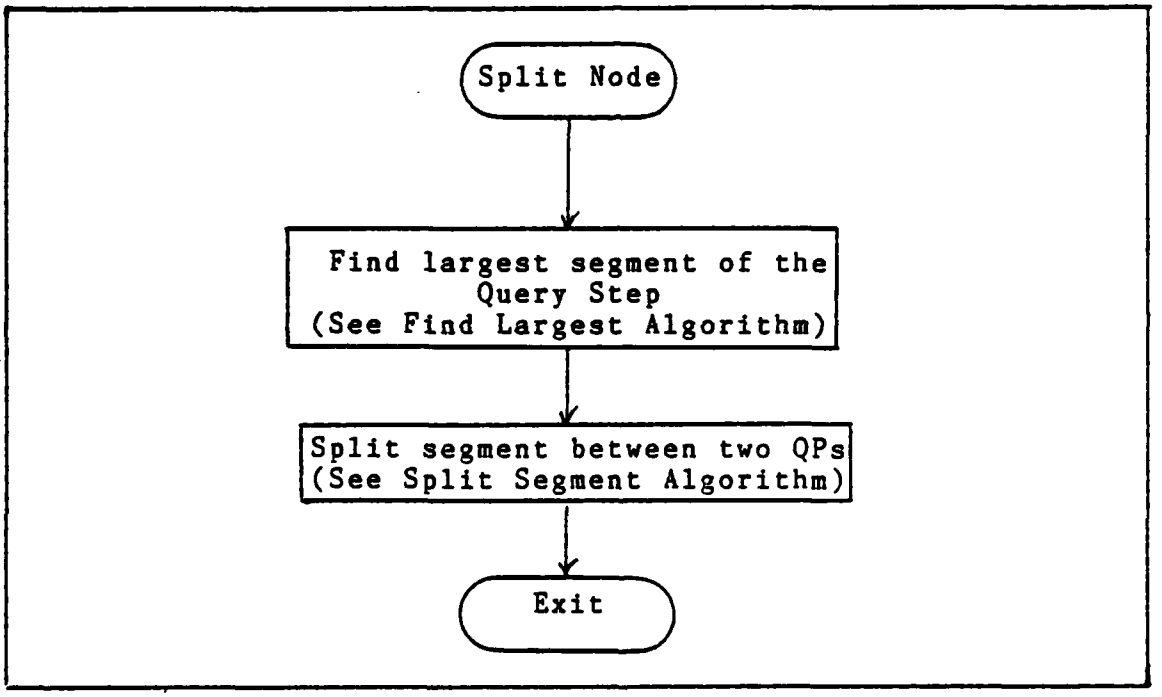


Figure 13. Node Splitting Algorithm.

To find the largest query step segment (See Figure 14), the BCP looks at each QP's query step pointer. If it is equal to the query step pointer being split, then the BCP

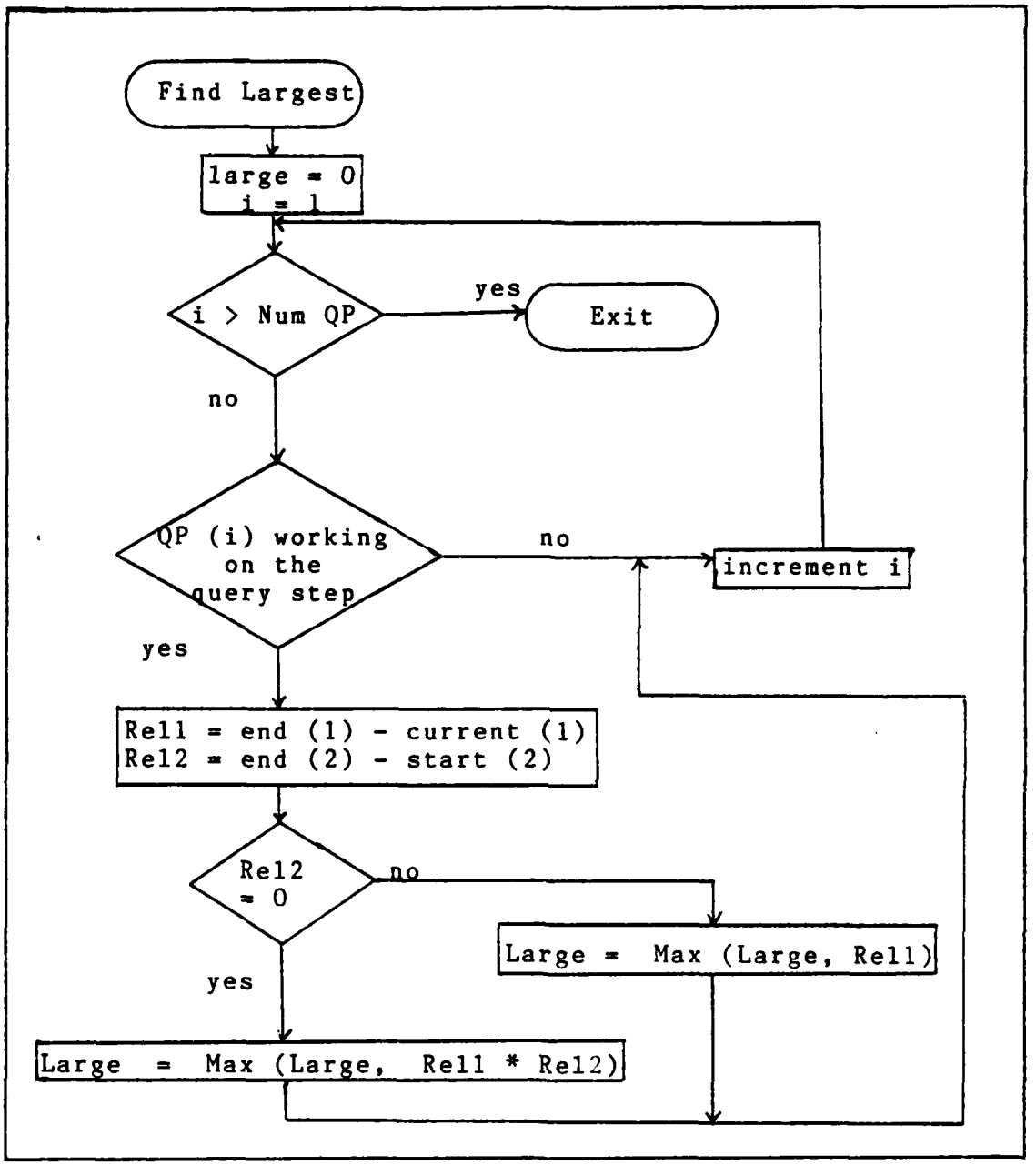


Figure 14. Find Largest Query Step Segment.

computes the number of pages left to process. For unary operations, this is the "end page" index less the "current page" index. For binary operations, this is the "end page" index less the "current page" index of the first relation, times the "end page" index minus the "start page" index of the second relation. The QP ("large QP") with the largest number of unprocessed pages will have its query step segment split with the free QP ("idle QP").

To split the "large QP" with the "idle QP" (See Figure 15), the BCP first chooses which input relation to divide. This will generally be the larger of the two relations (for the "diff" operation, it must be the first relation). Then the index values for each QP status field is set according to the algorithm shown in Figure 15.

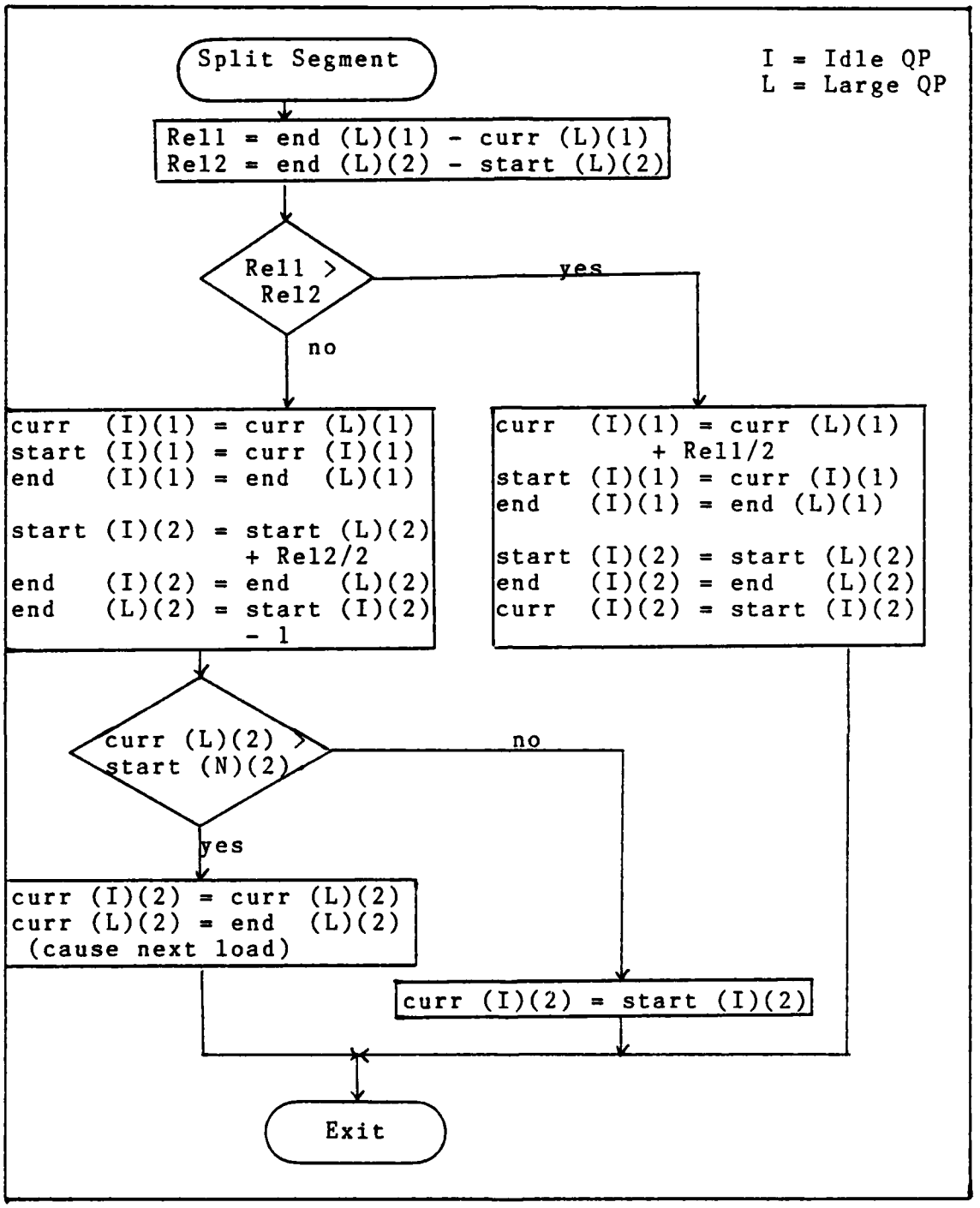


Figure 15. Split Node in Half Algorithm.

Node Splitting Example

Let query step 'X' be a "select" operation on a relation with 1000 pages. Let page size (m) be constant equal to one.

When the first QP is assigned to query step 'X', its starting page is set to zero, and the ending page is set to the number of pages in the relation.

```
status QP#1
     start (1) = 0;     current (1) = 0;     end (1) = 1000;
     start (2) = 0;     current (2) = 0;     end (2) = 0;
     page_size (1) = 1;        page_size (2) = 1;
     step = 'X';               file = 'X.0';
```

Note: File pages are numbered 0 to 999.

After QP#1 has completed 120 pages of the input relation, QP#2 ("new QP") becomes free and the BCP decides to assign it to query step 'X'. Since this is an additional QP, the node must be split. Because QP#1 has already completed the first 120 pages, its current status is:

```
status QP#1
     start (1) = 0;     current (1) = 120;    end (1) = 1000;
     start (2) = 0;     current (2) = 0;      end (2) = 0;
     page_size (1) = 1;        page_size (2) = 1;
     step = 'X';               file = 'X.0';
```

The BCP finds the "large QP" by searching all QPs already working on query step 'X' (in this case there is only one), and determines which has the largest number of pages to process. QP#1 has 880 (end (1) - start (1)) pages left. These pages are split between the "large QP" and the "new QP" with the "new QP" getting the latter half. The result is:

```
status QP#1
     start (1) = 0;     current (1) = 120;     end (1) = 560;
     start (2) = 0;     current (2) = 0;       end (2) = 0;
     page_size (1) = 1;        page_size (2) = 1;
     step = 'X';               file = 'X.0';

status QP#2
     start (1) = 560;   current (1) = 560;     end (1) = 1000;
     start (2) = 0;     current (2) = 0;       end (2) = 0;
     page_size (1) = 1;        page_size (2) = 1;
     step = 'X';               file = 'X.1';
```

Now a third QP becomes free. During the time, QP#1 completed

50 more pages, and QP#2 completed 80 pages. So

```
status QP#1 1st relation is:
     start (1) = 0;     current (1) = 170;       end (1) = 560;

pages left = 560 - 170 = 390

status QP#2 1st relation is:
     start = 560;      current (1) = 640;       end (1) = 1000;

pages left = 1000 - 640 = 360
```

So, QP#1 is "large QP" and QP#3 is "new QP". After splitting the pages in half the status of the system is:

```
status QP#1 1st relation and file:
     start (1) = 0;     current (1) = 170;       end (1) = 360;
     step = 'X';        file = 'X.0';

status QP#3; 1st relation and file:
     start (1) = 360;   current (1) = 360;       end (1) = 560;
     step = 'X';        file = 'X.2';

status QP#2; first relation and file:
     start (1) = 560;    current (1) = 640;     end (1) = 1000;
     step = 'X';        file = 'X.1';
```

This process of node splitting continues as additional QPs are assigned to query step 'X', and the output files are linked in ascending order based on starting page number. The resulting logical file is in the same sort order as the input file.

Locking Scheme

To insure data integrity, the database must lock each base relation before accessing the data. The BCP uses both "read" and "write" locks on the base relations. Temporary relations do not require locking because only their own query tree will access them. A "write" lock prevents any other query step from accessing the relation. A "read" lock prevents any updates on the relation.

During the QP assignment, the locks are checked on any

base relation the query step might access. The file locks are checked only the first time a QP is assigned to the query step. During "node splitting", the files are already locked by the query step and do not require additional locking. If the "write" lock is locked, or if the "read" lock is locked and the query step is an update operation, then the query step is ineligible for assignment

A retrieval operation increments the "read" lock when first assigned to a QP, and decrements the "read" lock upon completion. Likewise, an update operation increments and decrements the "write" lock.

This locking scheme is extremely simple, but inefficient. A high priority update query step may be locked out by previous jobs and skipped over. This would allow lower priority retrieval steps to place additional "read" lock on the file. The result is a lockout condition of a higher priority job.

An improved locking scheme could take advantage of the current update procedure. Update operations only read the original file and write to a different output file. Upon completion, it deletes the old file and renames the output file to the old input file name. This means that an update operation could lock a file that has the "read" lock set. Upon completion of the update, if the "read" lock is still set, rather than delete the file, the BCP would rename the file and have all query steps reading the data close their input files and reopen them with the new name. The last

query step to unlock the obsolete file would also delete it. This provides a higher concurrence rate and eliminates the lockout problem.

Error Handling

The BCP error recovery capabilities are non-existent. If the software is unable to allocate storage or if any other unexpected results occur, the BCP prints the current module name and an error message, and halts. This will be inadequate once the frontend is operational, and should be corrected.

VIII Conclusion and Recommendations

Overview

The major goal of this thesis was to provide a working model of the BCP. Unfortunately, the project has fallen short of this goal. The following areas were not completed:

* communications software was not implemented (currently, the BCP interacts with dummy modules that provide a trace of all I/O to the BCP)
* only the general paging algorithm was implemented
* file locking mechanism was not implemented
* BCP commands were not implemented

Despite these failings, advances were made in the overall design of the Backend Relational Database Management System. The thesis provided a functional requirements analysis of the BCP along with the software algorithms necessary to achieve these requirements. It refined some of the query step operations discussed by Rogers. It included the addition of the frontend processor to the backend system to improve system flexibility and modularity. Chapter 3 proves the feasibility of splitting queries across several processors. The thesis effort also produced a starting base for farther advances to the BCP.

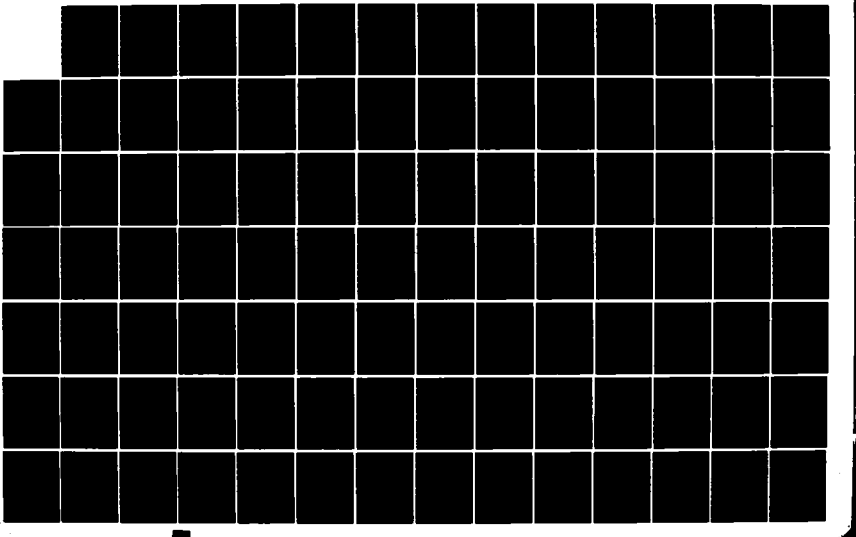
Suggested Advancements

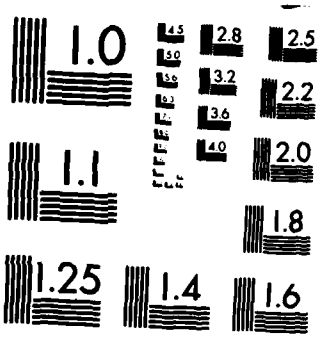
There is still a tremendous amount of work left to be completed on the backend database system. Initial work that can be done on the current version of the BCP include:

* completion of the paging algorithm for remaining query step operations (sort and merge paging algorithms)

AD-A151 892 BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/3

UNCLASSIFIED D M PONTIFF DEC 84 AFIT/GCS/ENG/84D-22 F/G 9/2 NL





* implementation of a locking strategy for database files
* design and implementation of the BCP commands (this would require additional data structures to manage partially completed query steps (i.e. query which have been preempted))
* implemention of BCP communication abilities (should be interupt driven)

Future enhancements to the BCP, once a common or shared memory device (MBU) is available, to improve system performance may include:

* design and implementation of an optimized QP assignment algorithm which supports pipelining
* design and implemention of a buffer allocation algorithm
* modification of QP and BCP so that a QP can be operating on two (or more) query steps concurrently (this would reduce QP idle time and facilitate pipelining)
* determination of optimum file structure (system currently assumes the use of simple flat files)

Short term advancements within the Backend Database System include:

* implementing Roth's DBMS on the FE, and providing the tree translation software needed to convert the Roth query tree to the BCP query tree
* design and implementation of the QPs and MSU

Long term goals of this thesis project remain unchanged from Fonden's original designs.

Parting Comments

This section will discuss some general comments about the developement of the BCP from the author's perpective. The use of SADT to determine system requirements was a tremendous help in the initial phase of the thesis. Several months of effort were spent discussing with Dr. Hartrum specifically what the BCP must provide, and what type of

support it could expect from the other components within the backend system. Once the major functions of the BCP were determined, the SADT became a liability because of the amount of time needed to modify them for minor changes within the system. After modifing the SADT diagrams several times, they were not updated for each change and the current diagrams provided in Appendix C are a mix of the designed BCP system, and the initial SADT requirement diagrams.

Once the requirements were fixed, the majority of the coding was completed in six weeks. Part of the reason the coding was completed so quickly was due to the SADT diagrams. Having fixed requirements, it was easy to implement and test the individual functions. Modular code was used to make it easy to modify one functional area without affecting the next one.

The language used to develop the BCP was BDS C. While C tends to be a cryptic language, it is very well suited for systems work. The BDS C is an impressive compiler. It is able to quickly determine if syntax errors exist in the code so the programmer is able to spend more time working, and less time waiting for compile runs (and error reports).

Appendix A:

Glossary

AFIT -- Air Force Institute of Technology

Attribute Field Identifier -- A field length and start location for fixed length relations. A zero value and a position location for varying length relations.

Attribute List -- A list of attribute field identifiers.

base relation -- a permanent relation created by the DBA and has all domains and attributes stored in the data dictionary.

BCP -- Backend Control Processor.

BDS C -- A C compiler designed to run on micro systems.

Binary Relational Operations -- A relational operations that act against two input files (i.e. product, join, union, etc.).

CP/M -- An operating system designed to run on micro systems.

DB -- Database.

DBA -- Database Administrator.

DBMS -- Database Management System.

DD -- Data Dictionary.

DDL -- Data Definition Language.

FE -- Frontend.

IMM -- Internal Memory Module.

Independent Parallelism -- The simultaneously processing of two (or more) parts of a query which will be joined at a later stage of the query tree.

LHS -- Left Hand Side of an equation.

MBU -- Memory Buffer Unit.

Modification List -- A list of attribute field identifiers followed by a new value to be stored in the fields.

MSU -- Mass Store Unit.

Node Splitting Parallelism -- Having several processors simultaneously act on different pages in the same query step.

Pipelining Parallelism -- Having the output of a process(es) being immediately fed into a second processor(s) to complete the next step of a query.

QP -- Query Processor.

QS -- Query Step.

Query -- A user request to access data within the database (either update or retrieval).

Query Step -- A single relational operation to be performed on relation.

Query Tree -- The combination of many Query Step needed to perform the requested query of a user.

Retrieval Request -- A query which only reads data from the database.

RHS -- Right Hand Side of an equation.

SADT -- Structure Analysis and Design Technique.

Selection Criterion -- A set of boolean (ANDs and ORs) conditions to allow the comparison of an attribute field value against a constant or different attribute field value.

Task Tree -- an ordered colection of Query Trees.

temporary relation -- An intermediate relation created to answer a user retrieval query.

Unary Relation Operation -- A relational operations that act against only one input file (i.e. project, select, count, etc.).

Update Request -- A query which reads and writes data in the database.

Appendix B:

Single Processor DBMS (SADT)

The high-level SADT diagrams of a single processor DBMS are included to provide a rough functional breakdown of a DBMS. This helped solve the problem of how to split the DBMS responsibilities between the FE and the BCP. The breakdown determined the major functions and logical break points.

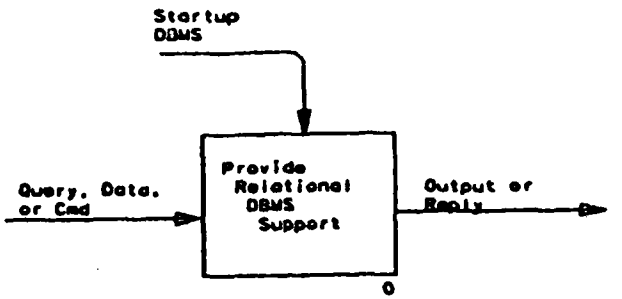
Since the QPs were designed for executing queries, and the BCP's major purpose was to control the QPs, the decision was that the FE would essentially become a single processor DBMS except for the actual execution of the queries. The optimized queries would be passed down to the BCP, and from here, down to the QPs.

A-0 Single Processor DBMS

Abstract: This is the environment node.

The system is a simple relational DBMS, it receives user queries, data, and commands. It acts on the input as a relational DBMS, and returns either a reply or an output relation.

AUTHOR: Capt. Dale M. Pontif
PROJECT: BACKEND
DATE: 11/24/84
REV:
WORKING DRAFT
X RECOMMENDED PUBLICATION
READER
DATE
CONTEXT:
NOTES: 1 2 3 4 5 6 7 8 9 10



NODE: DMPBCP/I0/A0.2
TITLE: Single Processor DBMS
NUMBER:

A0 Provide Relational DBMS

Abstract: This is the highest level within a DBMS, it receives input (either a query, data file, or command), acts against the input, and returns a reply or an output relation.

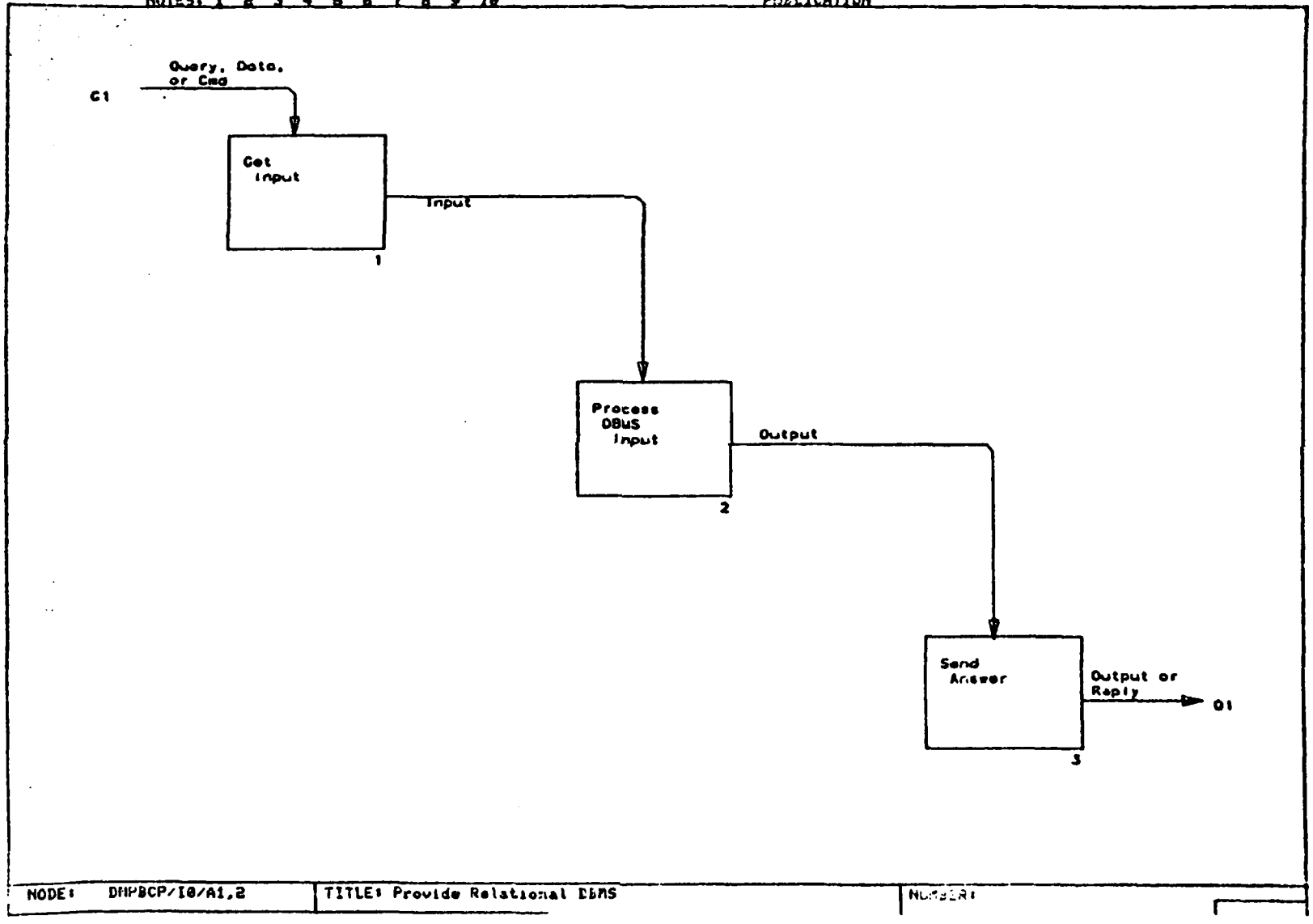
A1 Get Input is the ears for the DBMS. It listens for incoming user requests, and stores them in a form the rest of the DBMS can understand.

A2 Process DBMS Input is the body of the DBMS. It performs all the actual work done by the DBMS. It checks the syntax of user requests, verifies the user's access rights, logs the transaction, and executes the request.

A3 Send Answer is the mouth of the DBMS. It takes the results from the DBMS and converts them into a form that the user's process can understand.

AUTHOR:Capt. Dale M. Pontif
PROJECT:BACKEND
DATE:11/24/84
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

WORKING DRAFT
X RECOMMENDED PUBLICATION
READER
DATE
CONTEXT:



NODE: DMPBCP/I0/A1,2
TITLE: Provide Relational DBMS
NUMBER:

A2 Process DBMS Input

Abstract: This is the body of the DBMS. It performs all the actual work done by the DBMS. It checks the syntax of user requests, verifies the user's access rights, logs the transaction, and executes the request.

A21 Analyze Syntax checks syntax for all queries, data input files, and commands. It verifies if the relations and/or fields exist within the database (this includes checking the data types).

A22 Verify Access provides data security checks. It verifies that the user has access rights to the relations, fields, or commands he is attempting to access.

A23 Log Transaction provides a transaction log of all queries and commands run against the database. The log is used for both backup and security purposes.

A24 Optimize Query arranges a complex query into a relatively efficient query form (tree).

A25 Execute Query actually accesses the relations and fields to provide the results requested by the user query.

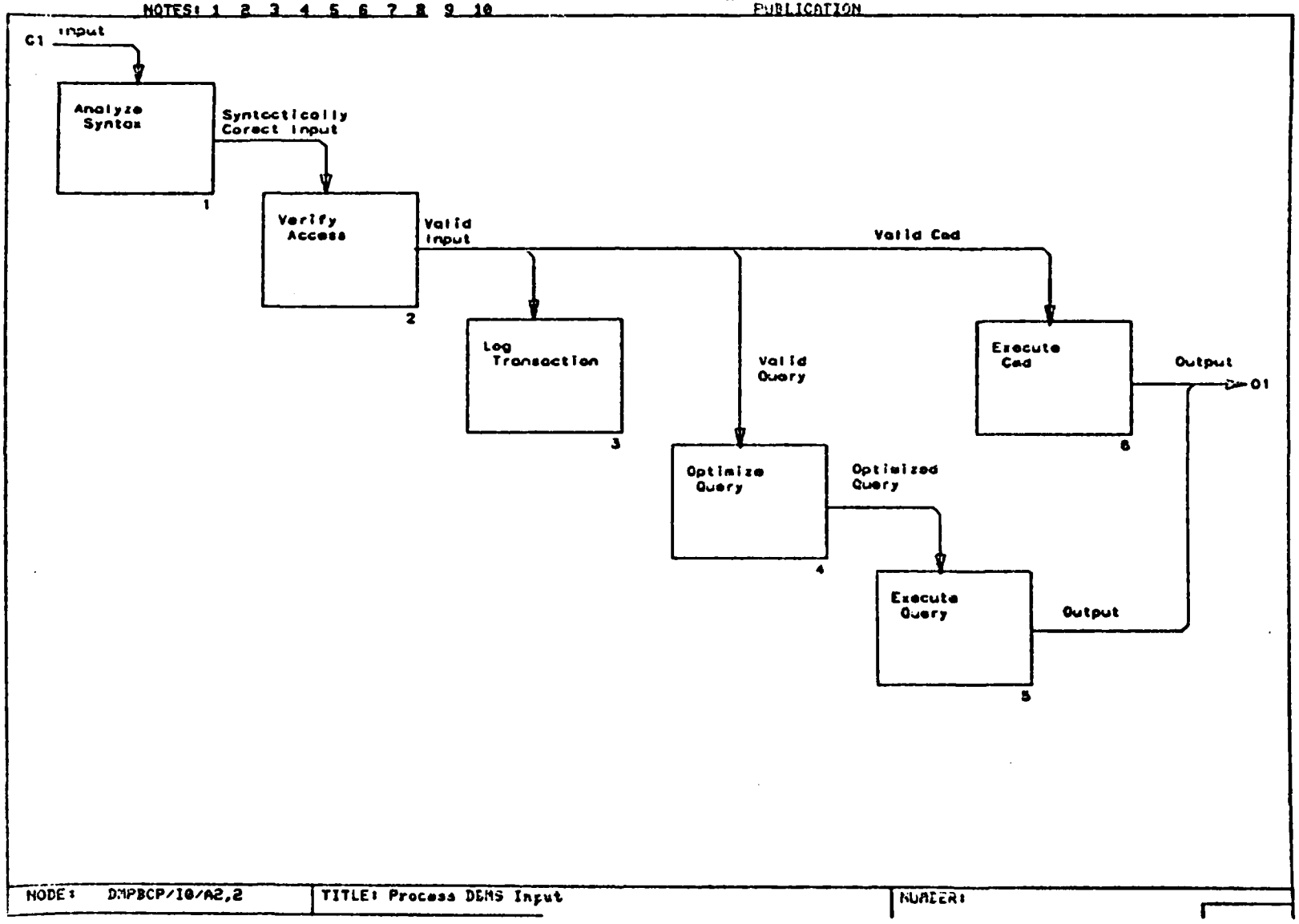
A26 Execute Cmd actually accesses the database tables and/or data dictionary to modify the database system.

AUTHOR:Capt Dale A. Pontiff
PROJECT:BACKEND
DATE:11/24/84
REV:

WORKING DRAFT
X RECOMMENDED PUBLICATION

READER DATE CONTEXT:

NOTES: 1 2 3 4 5 6 7 8 9 10



NODE: DMPBCP/I0/A2,2

TITLE: Process DEMS Input

NUMBER:

Appendix C:

Requirements Analysis of the Backend DBMS

```
A-0  Multi-Processor Backend Relational DBMS
A0   Provide Relational DBMS Support
A1       Initialize Database System
A2       Provide DBMS Functions
A21           Provide Frontend DBMS Functions
A212               Execute FE DBMS Functions
A2122                   Execute Preliminary DBMS Functions
A22           Provide BCP Functions
A222               Execute BCP DBMS Functions
A2223                   Add to Tak Tree
A2224                   Manage QP Assignment/Release
A2225                   Manage Active Query Steps
A2226                   Update Task Tree
A3     Shutdown System
```

A-0 Multi-Processor Backend Relational DBMS

Abstract: This is the environment node.

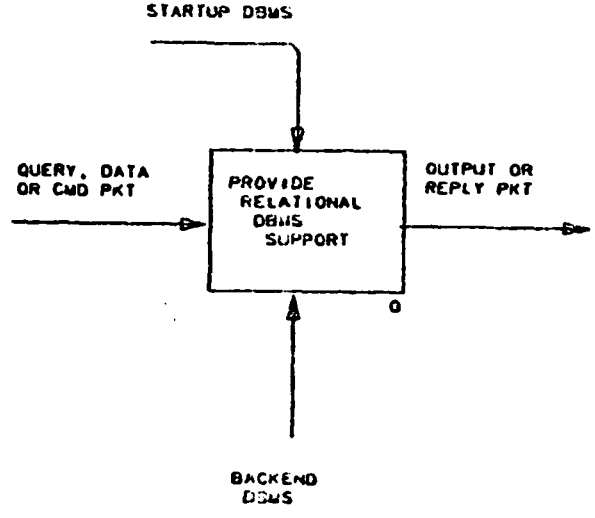
At this level, the Backend DBMS is seen as a Relational Database Management System. Once the Backend is activited (DBMS startup), it receives queries (both retrievals and updates), new data, and commands from a network, host system, or a CRT terminal. It responses with either an output relation (for retrievals) or a reply message for updates and commands (assuming nothing went wrong).

Packets are used to facilitate communications between processors.

AUTHOR:Capt Dale M. Pontiff
PROJECT:BACKEND
DATE:11/27.84
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

WORKING DRAFT
X RECOMMENDED PUBLICATION

READER DATE CONTEXT:



[illegible]PSCP/I0/0.1	TITLE: Multi-Processor Backend Relational DBMS

A0 Provide Relational DBMS Support

Abstract: This shows a simple breakdown of the DBMS. It contains startup, active, and shutdown phases.

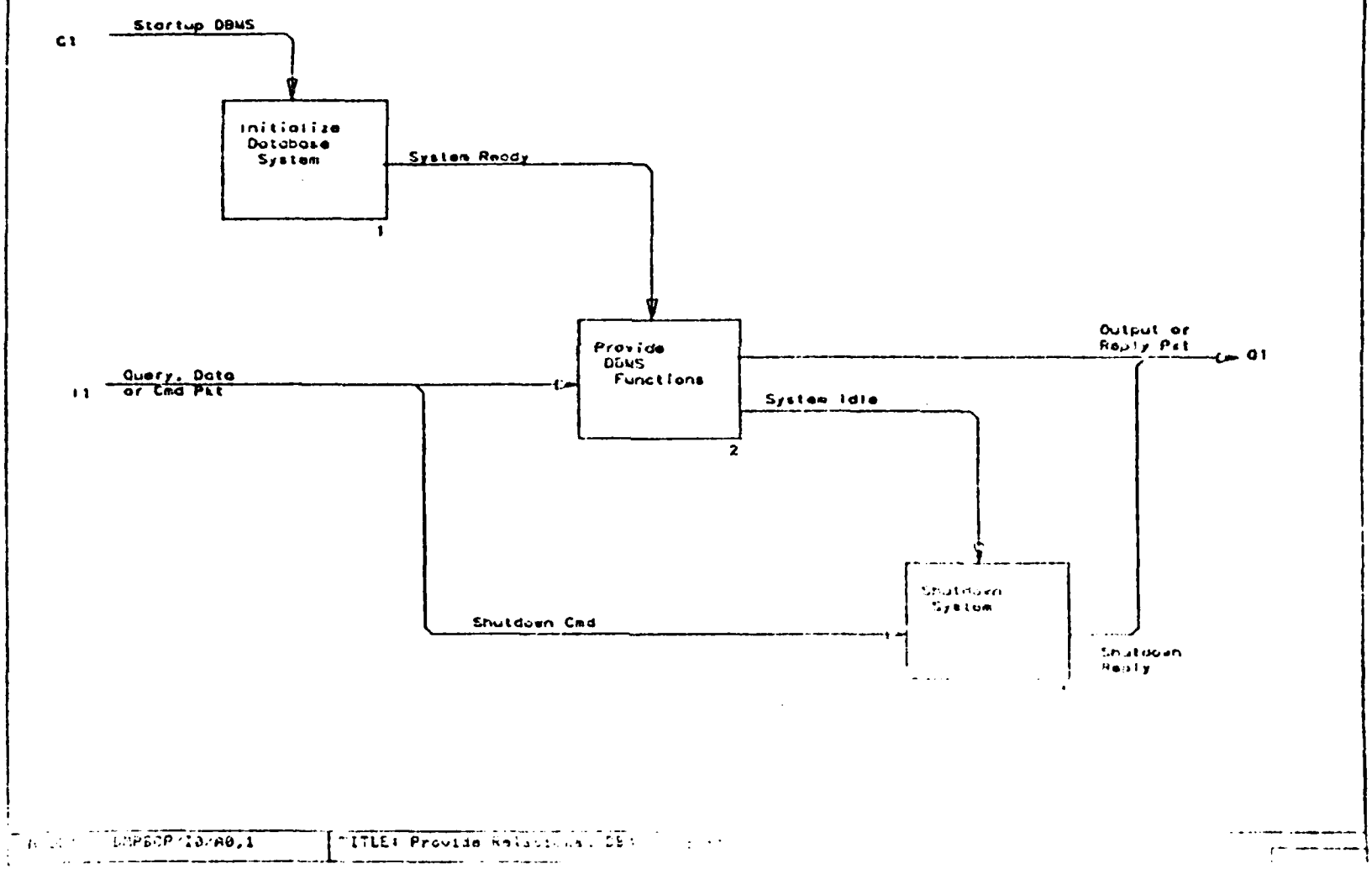
A portion of the startup phase will be initiated by a human operator. Once the system is up, it will provide database management for the existing database. The DBMS functions include relational operations (select, project, join, product, union, difference, intersect), update operations (insert, delete, modify), and miscellaneous operations (min, max, count, sort, sum). The DBMS commands provide some external control over the Backend system, and allow the DBA to modify the database data dictionary. The commands include: startup, shutdown, start job, stop job, abort job, change priority, job status, and DDL commands. The shutdown causes the system to stop accepting input, but should allow current queries to complete successfully.

A1 Initialize Database System entails all necessary steps needed to bring the DBMS up as a functioning unit (i.e. supply power, load OS, initialize system tables, etc.).

A2 Provide DBMS Functions contains the necessary DBMS functions needed to allow queries and commands to be levied against the existing DB.

A3 Shutdown System provides a safe, orderly method of terminating the operations of the DBMS.

AUTHOR:Capt Dale M. Pontiff DATE:11/27/84 WORKING DRAFT READER DATE CONTEXT:
PROJECT:BACKEND REV:
NOTES: 1 2 3 4 5 6 7 8 9 10 X RECOMMENDED PUBLICATION



DMPSCP-13-A0,1 TITLE: Provide Relational DBS

A1 Initialize Database System

Abstract: Entails all necessary steps needed to bring the DBMS up as a functioning unit (i.e. supply power, load OS, initialize system tables, etc.).

A large portion of the system startup may entail human intervention to power up the system and manually load the Operating Systems.

A11 Startup Frontend causes the frontend to be booted with the FE Operating System.

A12 Init Frontend causes the FE to initialize system tables and verify that the BCP is available for use.

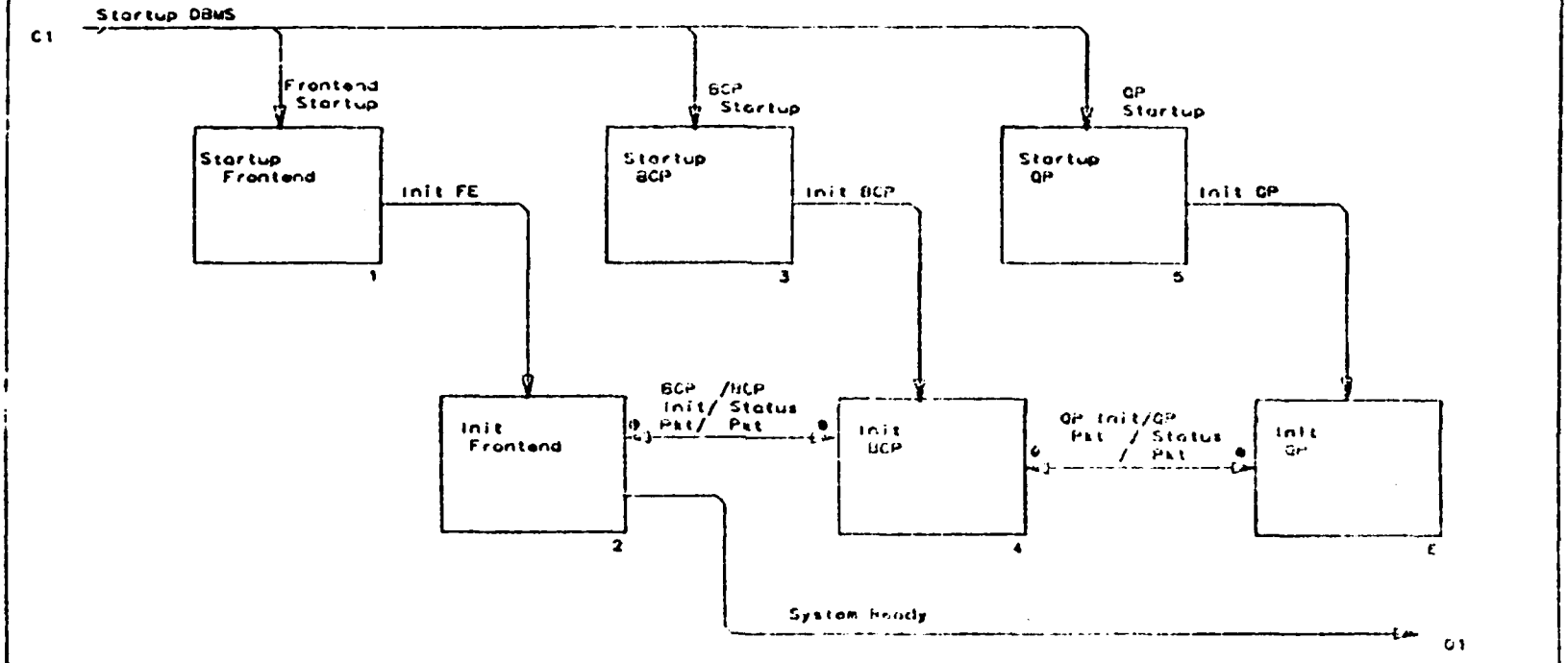
A13 Startup BCP causes the BCP to be booted with the BCP Operating System.

A14 Init BCP causes the BCP to initialize system tables and verify that at least one QP is available for use. Sends a message to the FE after initialization is complete.

A15 Startup QP causes the QPs to be booted.

A16 Init QP causes the QP to initialize any internal fields or tables. Sends a message to the BCP upon completion.

AUTHOR:Capt Dale N. Pontiff DATE:11/27/84 WORKING SER DATE CONTEXT:
PROJECT:BACKEND REV: SHEET
RECOMMENDED
NOTES: 1 2 3 4 5 6 7 8 9 10 PUBLICATION:



104

NAME: DPBSCP/IO A1.1 | TITLE: Initialize

A2 Provide DBMS Functions

Abstract: Contains the necessary DBMS functions needed to allow queries and commands to be levied against the existing DB.

The system is decomposed along "functional" lines of the major architecture components of the system. The frontend receives queries and commands from the outside, places the valid data into an optimized query tree and passes it on to the BCP. The BCP then decides which task in the tree to perform in which order, and assigns one or more QPs to work on a task. The mass storage unit (MSU) allows fast access to the DB pages and can transfer data into and out of the memory buffer unit (MBU) quickly. The memory buffers provide a (hopefully common) memory space in which the QPs can manipulate the data stored in the system. The QPs actually execute all the necessary relational functions against the data stored in the memory buffers.

A21 Provides Frontend DBMS Functions is responsible for communications between the backend system and the outside world. It is also responsible for most of the database management functions not directly related to the relational operations against the database (such as syntax checks, security checks, transaction log, and query optimization).

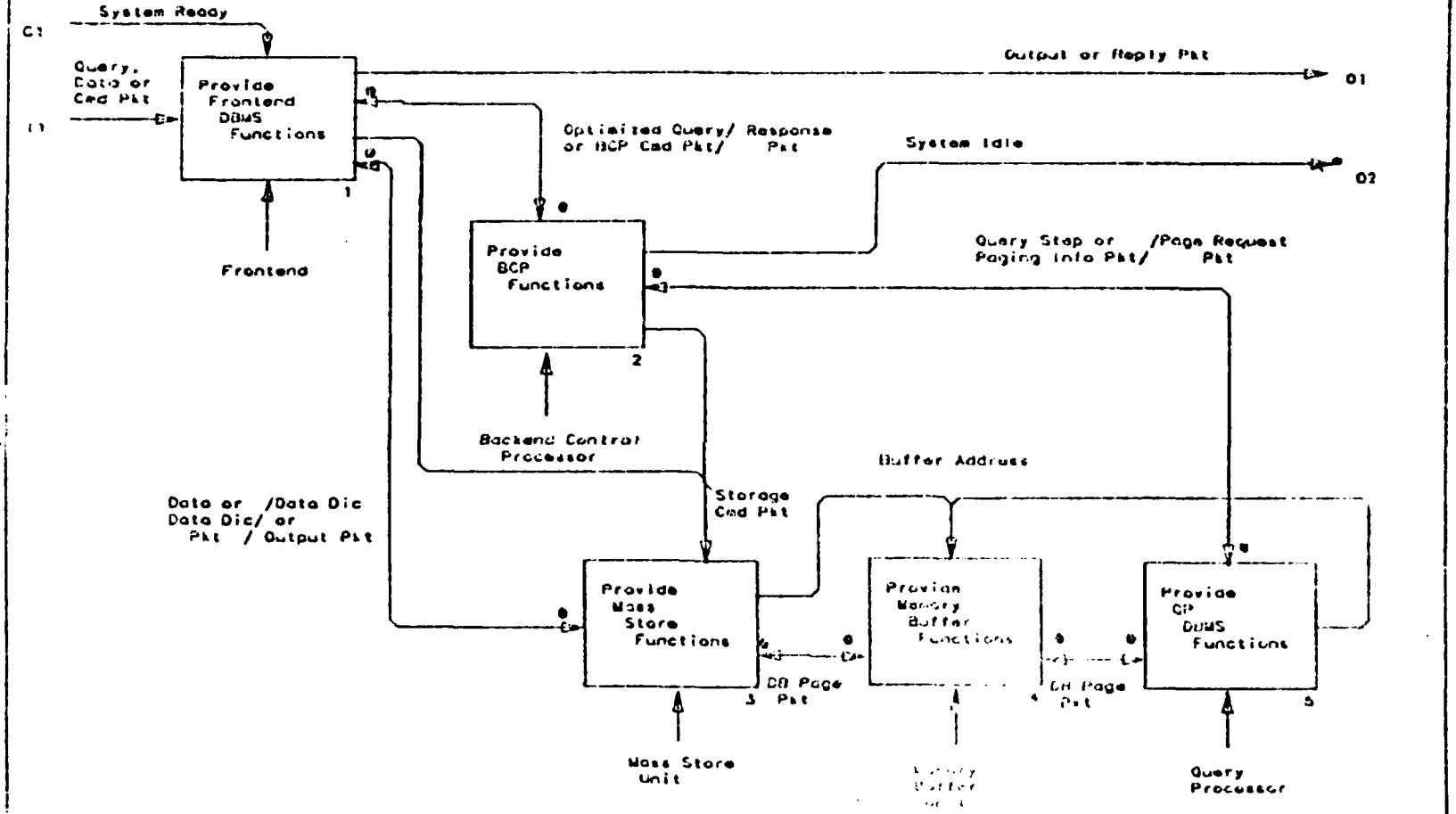
A22 Provide BCP Functions is responsible for scheduling query tasks and managing the system paging.

A23 Provide Mass Storage Functions is responsible for file management. It provides permanent storage of the existing database, plus temporary storage of any intermediate relations created during a query.

A24 Provide Memory Buffer Functions provides very fast scratch pad memory for the QPs to manipulate data.

A25 Provide QP DBMS Functions provides the relational operations (select, project, join, product, union, difference, intersection), update operations (insert, delete, modify) and micellaneous operations (min, max, count, sort, sum) that actual act on the data within the DBMS.

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REVI
WORKING
DRAFT
X RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
NOTES: 1 2 3 4 5 6 7 8 9 10



106

NODE: CMPBE/I0/A2.1
TITLE: Provide DBMS Functions

A21 Provide Frontend Functions

Abstract: The frontend (FE) is responsible for communications between the backend system and the outside world. It is also responsible for most of the database management functions not directly related to relational operations against the database (such as syntax checks, security checks, transaction log, and query optimization).

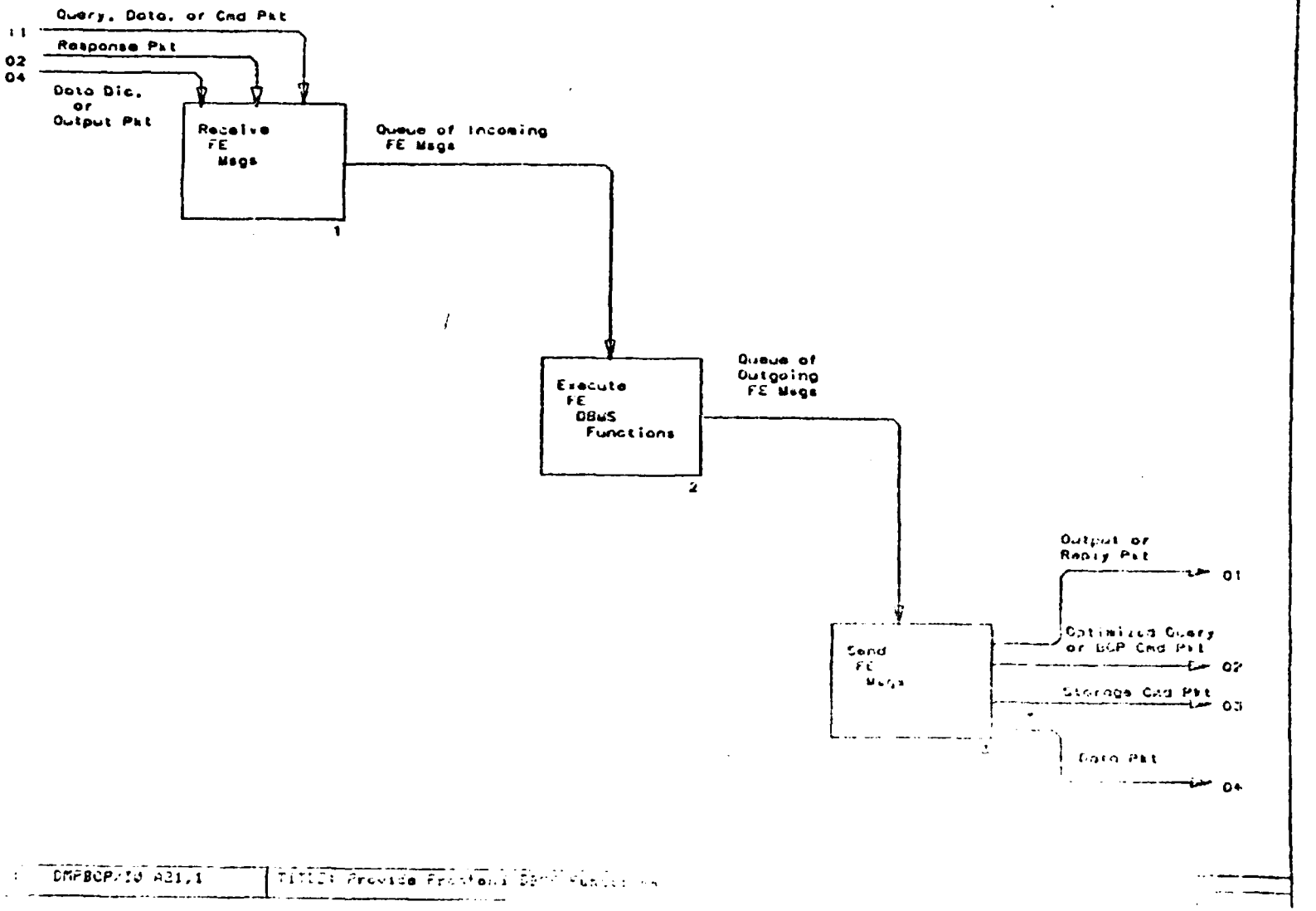
The FE brings request from the outside world into the backend. It performs the preliminary data checks on the information before passing the data to the mass store unit or the Query/Command on to the BCP. After the BCP has answered the query, it sends a response back to the FE telling it that it has completed the query and where the resulting relation is stored on file. The FE then request the MSU to send it the output which it forwards to the outside world.

A211 Receive FE Msgs listens for incoming messages from the host system or from other components of the Backend System. The messages are converted into a useable form for the FE.

A212 Execute FE DBMS Functions acts on incoming messages from the other processors. Its major functions are; to receive and validate queries/commands from the host, manage the Data Dictionary, pass the queries down to the BCP, and send replies back to the host.

A213 Send FE Msgs converts the internal system structures into a form that can be transferred to the other processors.

AUTHOR: Capt Dale N. Pontiff DATE: 11/27/84 WORKING DRAFT READER DATE CONTEXT:
PROJECT: BACKEND REV: X RECOMMENDED PUBLICATION
NOTES: 1 2 3 4 5 6 7 8 9 10



108

DMFBCP.30 A31.1 | TITLE: Provide Front-end DBMS Functions

A212 Execute FE DBMS Functions

Abstract: Acts on incoming messages from the other processors. Its major functions are; to receive and validate queries/commands from the host, manage the Data Dictionary, pass the queries down to the BCP, and send replies back to the host.

The FE looks at the top message in the incoming queue and determines what action it should take. If the message is from the host system, the FE validates the query/command. If it was a DDL command, it modifies the Data Dictionary as needed. If the input message was a response to a previous query/command, then the FE builds a user reply. All messages which must be sent to the other processors are then queued and sent at the FE's earliest opportunity.

A2121 Determine FE Action reads the top message in the queue and calls the correct module for that message type.

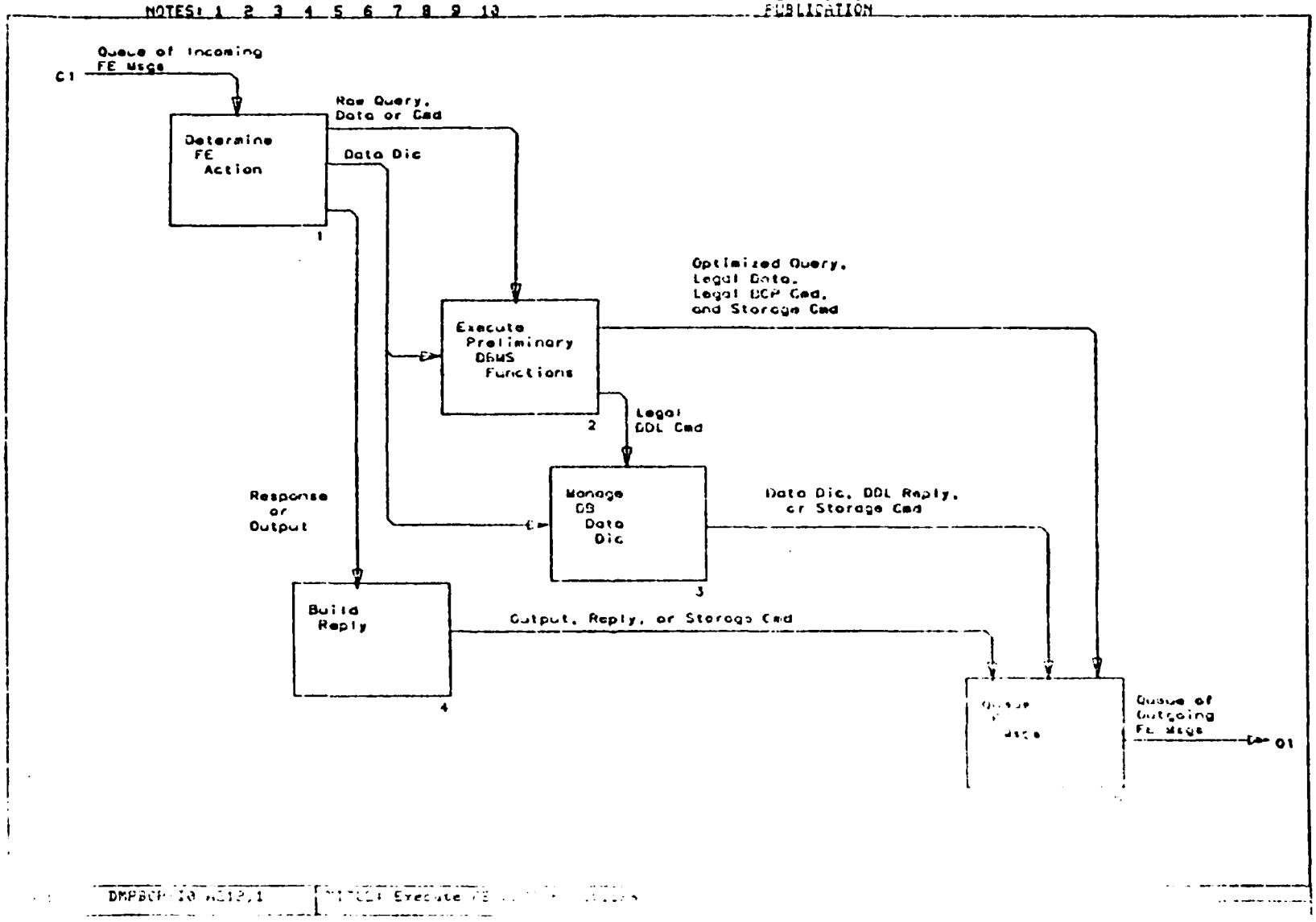
A2122 Execute Preliminary DBMS Functions receives raw input from the host system. It checks the syntax and user access rights, and logs the transactions. If it is a retrieval query, it optimizes the query tree.

A2123 Manage DB Data Dic is responsible for maintaining the database data dictionary. If the input message was a valid DDL command, it modifies the Data Dictionary as needed.

A2124 Build Reply receive responses from the BCP and Output relations from the MSU and formats the data to be forwarded to the host system.

A2125 Queue FE Msgs places any outgoing communication messages in a queue.

AUTHOR:Capt Dale M. Pontiff DATE:11/27/84 ORIGINAL READER DATE CONTEXT:
PROJECT:BACKEND REV1 DRAFT
X RECOMMENDED
NOTES: 1 2 3 4 5 6 7 8 9 10 PUBLICATION



110

A2122 Execute Preliminary DBMS Functions

Abstract: Receives raw input from the host system. It checks the syntax and user access rights, and logs the transactions. If it is a retrieval query, it optimizes the query tree.

The frontend performs any data checks and manipulations that are not relational in nature. This includes the following four major functions, but other minor functions may be added when their need is discovered.

A21221 Analyze Syntax checks syntax for all commands and queries. It also verifies if the relations and/or fields requested exist in the database.

A21222 Verify Access provides data security checks. It verifies if the user has access rights to the relations, fields, or commands he is attempting to access.

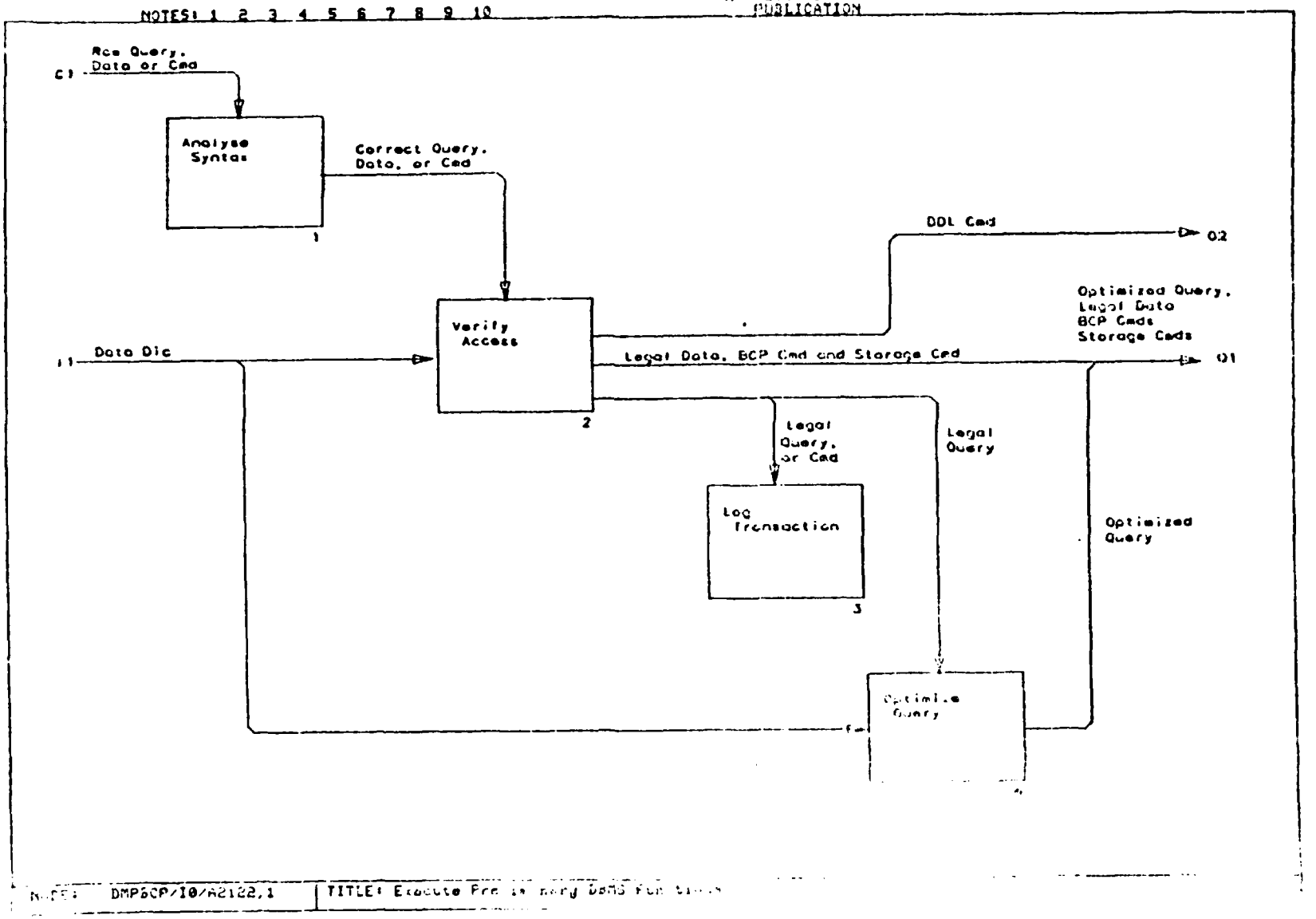
A21223 Log Transaction provides a transaction log of all queries and commands (both for backup and security purposes).

A21224 Optimize Query arranges a complex query into a relatively efficient query form (tree).

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REV:

	WORKING DRAFT	READER	DATE	CONTEXT:
X	RECOMMENDED PUBLICATION			

NOTES: 1 2 3 4 5 6 7 8 9 10



NOTES: DMPSCP/I0/A2122,1 | TITLE: Execute Pre in Query DBMS For [illegible]

A22 Provide BCP Functions

Abstract: The BCP is responsible for scheduling query tasks and managing the system paging.

The BCP receives queries and commands from the FE, and page requests from the QPs. These messages are queued and handled one at a time by the BCP. Commands are executed in the BCP, while queries are placed in the task tree to be scheduled as QPs become free. Page requests from the QPs cause the BCP to have the MSU page data into and out of the MBU.

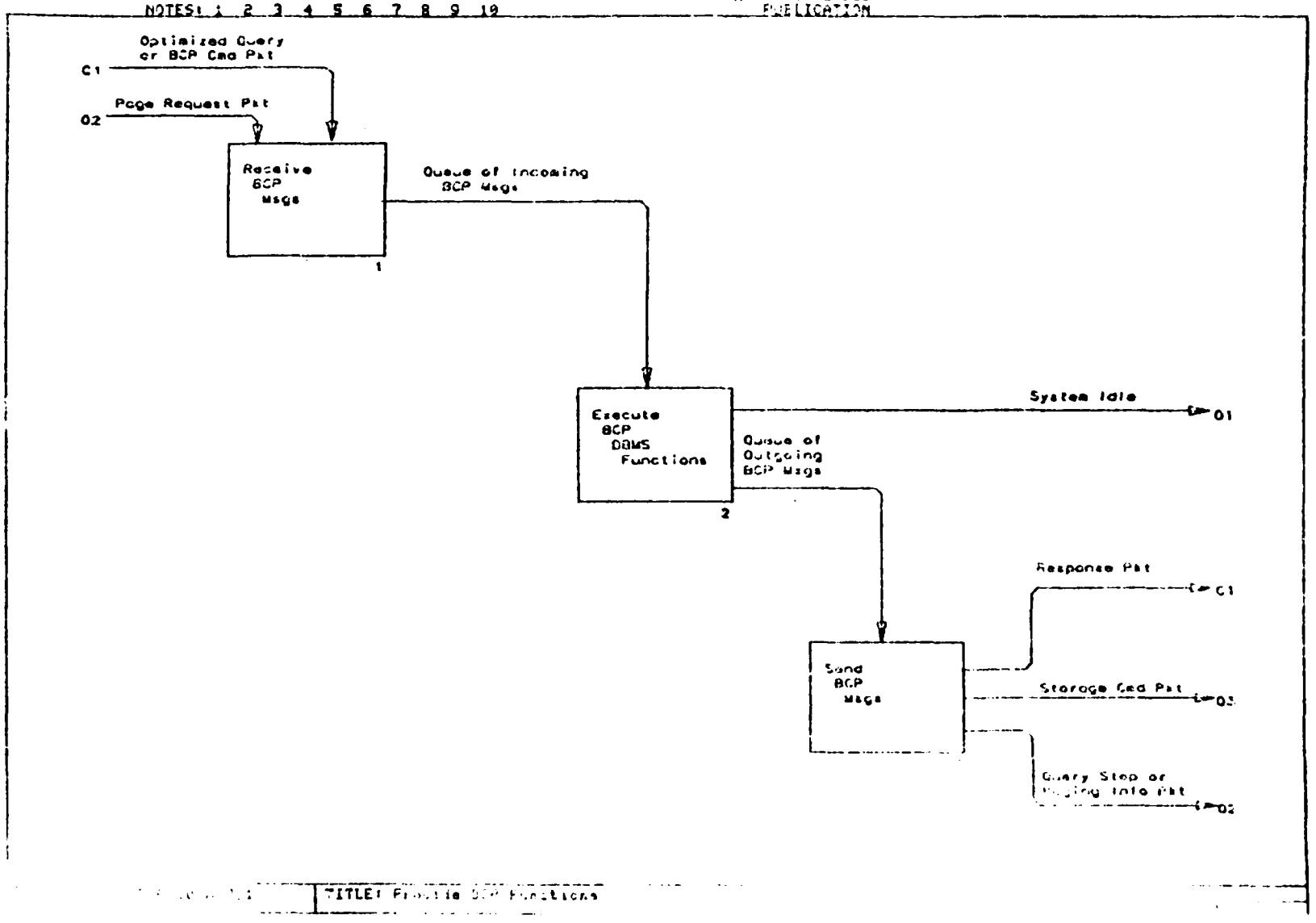
A221 Receive BCP Msgs listens for incoming messages from the frontend or the query processors.

A222 Execute BCP DBMS Functions executes BCP commands, schedules query step to QPs, and manages the paging algorithm of each operation.

A223 Send BCP Msgs converts the BCP's internal data structures into a form that can be transferred to the other processors. It sends responses to the FE, storage commands to the MSU, and query steps and paging information to the QPs.

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

WORKING DRAFT | READER | DATE | CONTEXT:
X RECOMMENDED PUBLICATION



TITLE: Process BCP Functions

A222 Execute BCP DBMS Functions

Abstract: Executes BCP commands, schedules query steps to the QPs, and manages the paging algorithm of each operation.

This looks at the message at the top of the input queue and determines what type of action should be taken. Commands are executed by the BCP and affect only a specific job. Certain commands (stop/abort job) may cause a preemption to be signaled against a running query.

Incoming queries are added to the existing Task Tree according to job priority. Whenever there is a free QP, the BCP examines the Task Tree, File Status, and System Resources Status to determine which Query Step should be run next. It passes the Query Step down to the active query step module.

Page request messages from the QPs are handled by the paging system, which supplies the next page needed to complete the task. Upon the completion of a Query Step, certain cleanup operations may be necessary. These include, checking for the completion of a query, removal of the Query Step node, and any old intermediate relations.

A2221 Determine BCP Action reads the message at the top of the queue and takes the appropriate action.

A2222 Execute Cmd allows some external job control commands to affect the system job scheduler.

A2223 Add to Task Tree phases the new query into the task tree according to its priority.

A2224 Manage QP Assignment/Release selects the next Query Step to be executed, and determines which Query Steps must be preempted when a job is stopped.

A2225 Manage Active Query Steps directs the QPs and controls the system paging of the MBU and MSU.

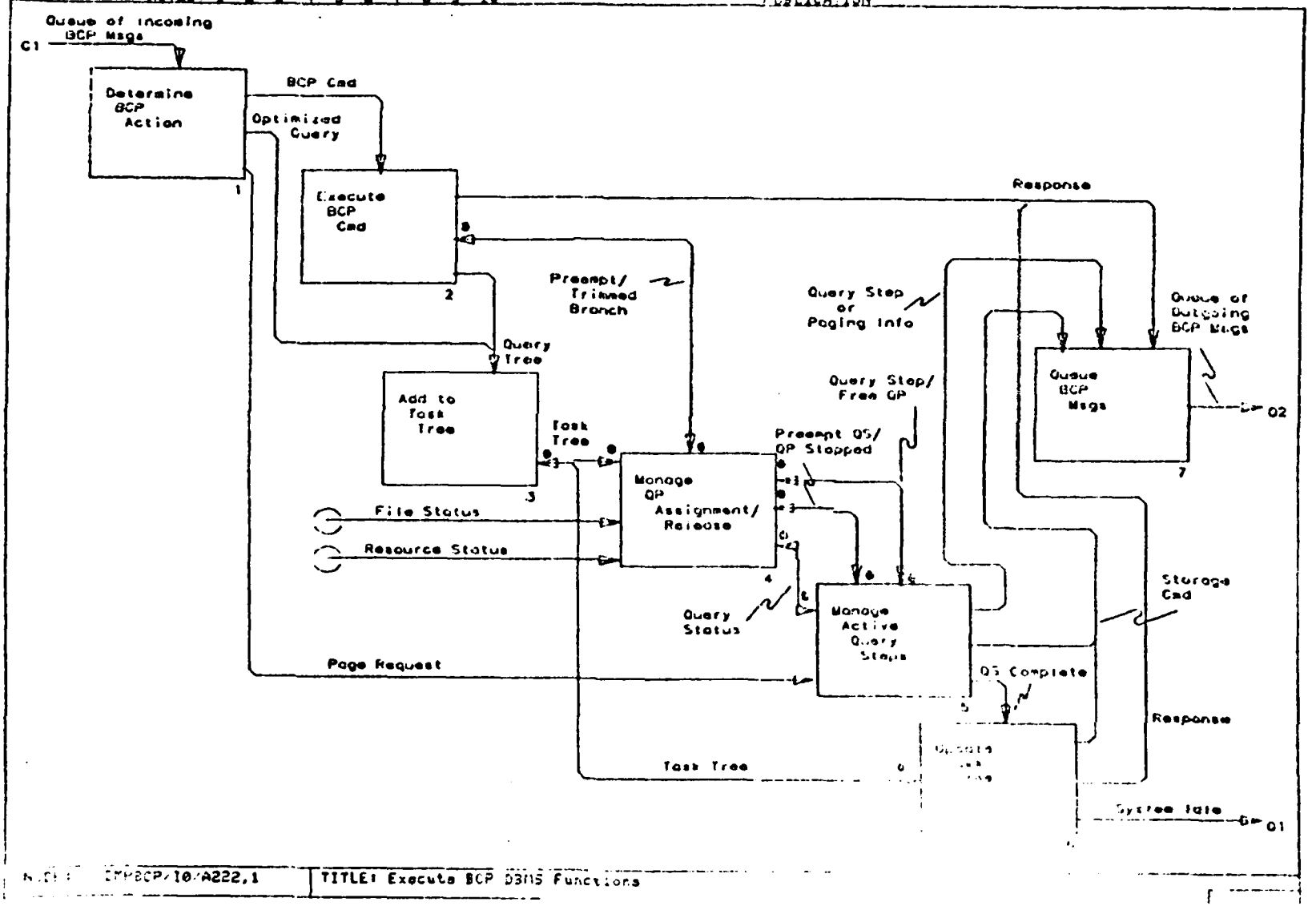
A2226 Update Task Tree checks for query completion, removes old information, and deletes the query from the task tree.

A2227 Queue BCP Msgs places any outgoing communication messages in a queue.

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REVI

	WORKING DRAFT	READER	DATE	CONTEXT:
X	RECOMMENDED PUBLICATION			

NOTES: 1 2 3 4 5 6 7 8 9 10



NAME: EMPBCP/10/A222,1 | TITLE: Execute BCP DBMS Functions

A2223 Add to Task Tree

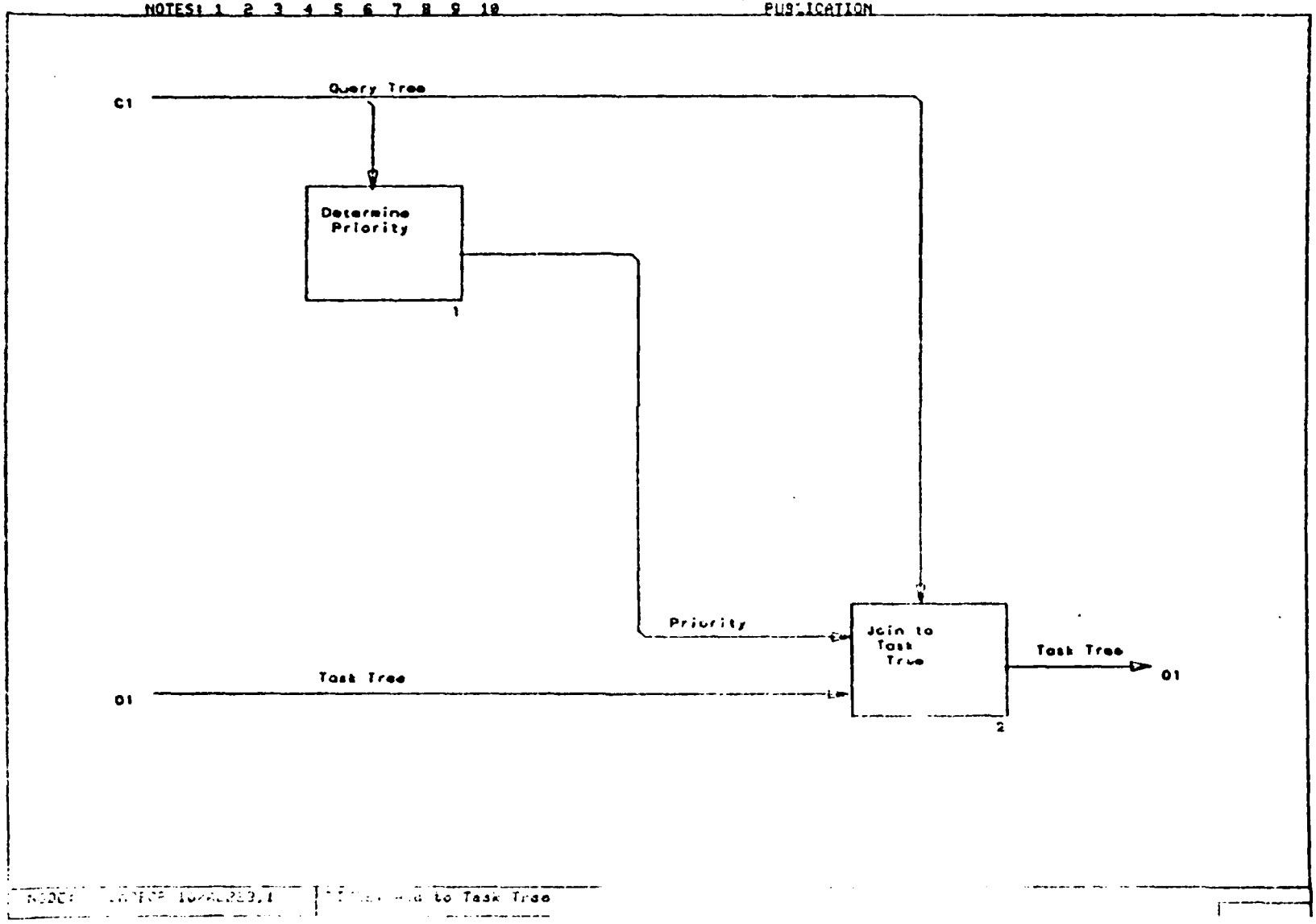
Abstract: Phases the new query into the task tree according to its priority.

A22231 Determine Priority generates a priority rating for a query.

A22232 Join to Task Tree adds the incoming query tree into the task tree as another branch, The query tree will be joined at the root node based on its priority.

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REV:
NOTES: 1 2 3 4 5 6 7 8 9 10

	CONTEXT:
WORKING DRAFT	
X RECOMMENDED PUBLICATION	



A2224 Manage QP Assignment/Release

Abstract: Selects the next Query Step to be executed, and determines which Query Steps must be preempted when a job is stopped.

The system selects a leaf node of the highest priority job. It then decides if a QP should be assigned to operate on this node based on:

1) the number of QPs already acting against the node
2) the size of the file
3) the number of QPs available in the system
4) the number of leaf nodes in the task tree

If an additional QP is assigned to a node, then a compression node must be created.

The other two modules are used to halt a running job. If a job is preempted, any QSs being operated on must be stopped. After stopping the QPs, the job is removed from the task tree.

A22241 Select Highest Available Leaf chooses the leaf node with the highest priority. If it is directed to reselect, it ignores all previously selected leaves.

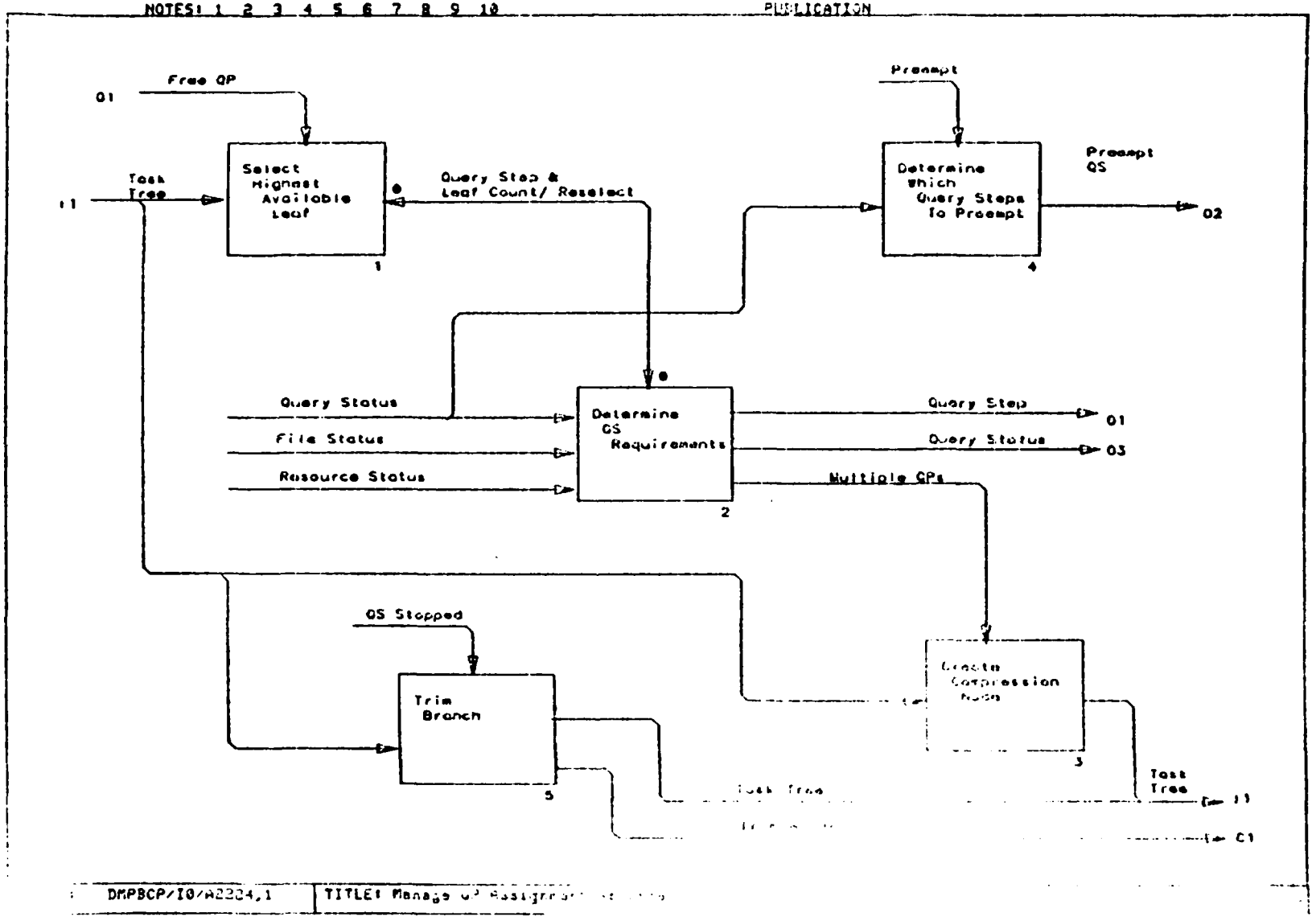
A22242 Determine QS Requirements decides if a QP should be allocated to work on this query step based on system status (see above).

A22243 Create Compression Node adds a compression node above a node that was split between two or more QPs.

A22244 Determine Which QS to Preempt is used to stop/abort a job. It causes any QPs working on the terminated job to stop after the completion of the current input page.

A22245 Trim Tree removes the query job from the task tree.

AUTHOR:Capt Dale M. Pontiff DATE:11/27/84 WORKING DRAFT READER DATE CONTEXT:
PROJECT:BACKEND REV: X RECOMMENDED PUBLICATION
NOTES: 1 2 3 4 5 6 7 8 9 10



DMPBCP/I0/A2224.1	TITLE: Manage [illegible]

A2225 Manage Active Query Steps

Abstract: Directs the QPs and controls the system paging of the MBU and MSU.

Upon receiving a Query Step, it is now considered active, and the Query Step is sent to the QP and allocated memory space in the MBU. The paging module is then told to load the initial input pages. Once a query step is active, the paging module will handle any additional page requests required by the QP. Upon paging out the last output page of a Query Step, that step is completed. The QP is cleared and marked free.

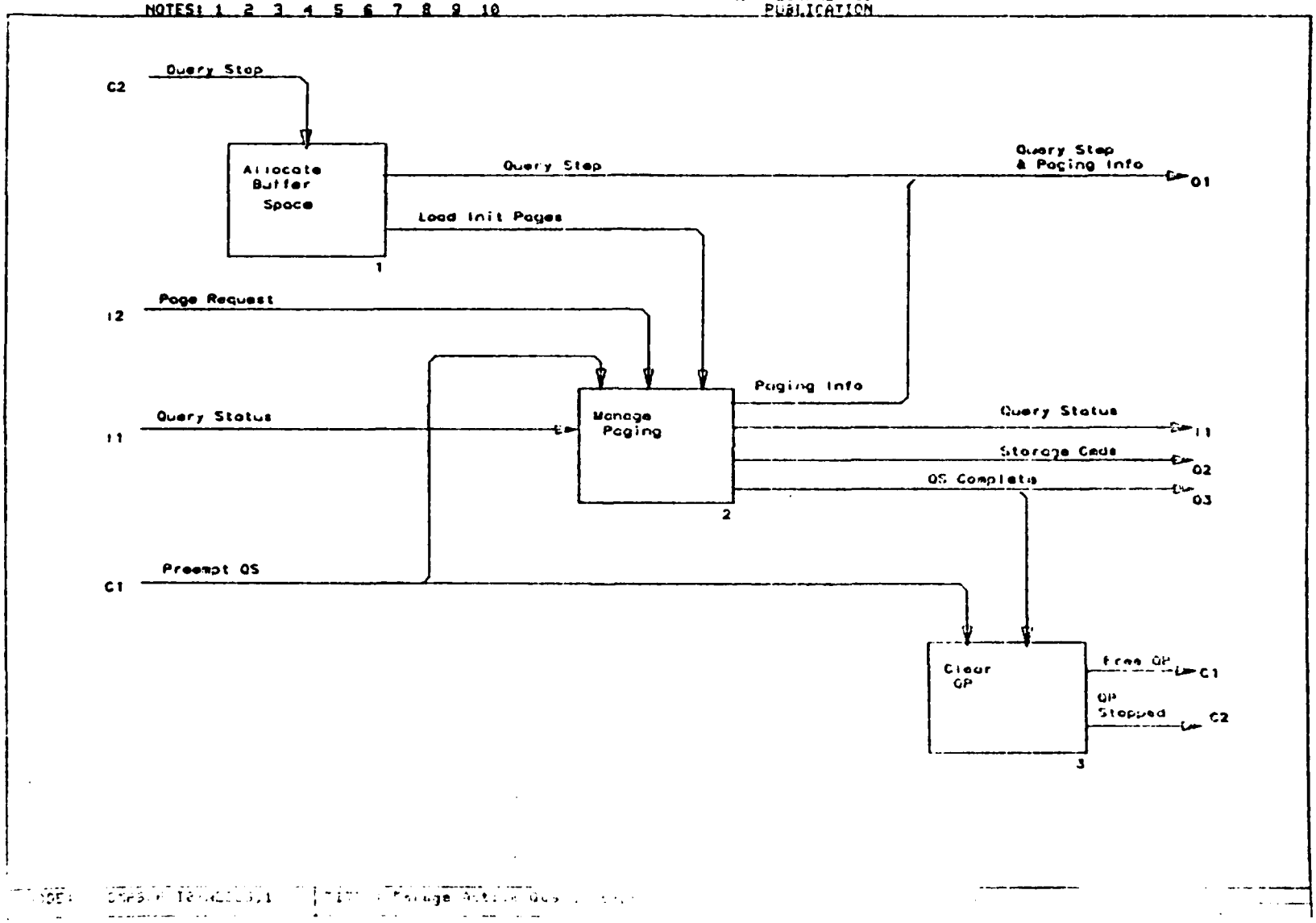
If a preempt QS is sent to the paging module, it stops suppling input pages to the QP.

A22251 Allocate Buffer Space determines how much buffer spaces is needed (and available) by the Query Step. It then forwards the Query Step down to the QP, and causes the paging system to load the initial pages of the relation.

A22252 Manage Paging handles the system paging algorithms.

A22253 Clear QP cleans up the QP and prepares it to receive a new Query Step.

AUTHOR:Capt Dale N. Pontiff DATE:11/27/84 WORKING DRAFT READER DATE CONTEXT:
PROJECT:BACKEND REV: x RECOMMENDED PUBLICATION
NOTES: 1 2 3 4 5 6 7 8 9 10



A2226 Update Task Tree

Abstract: Checks for query completion, removes old information, and deletes the query from the task tree.

Upon the completion of a QS, it removes the QS from the task tree, removes any old intermediate relations, and checks for the completion of the query.

A22261 Remove From Task Tree removes the QS from the Task tree.

A22262 Determine Query Completion Status decides if the query is completed. If so, it tells the frontend the location of any answer relation.

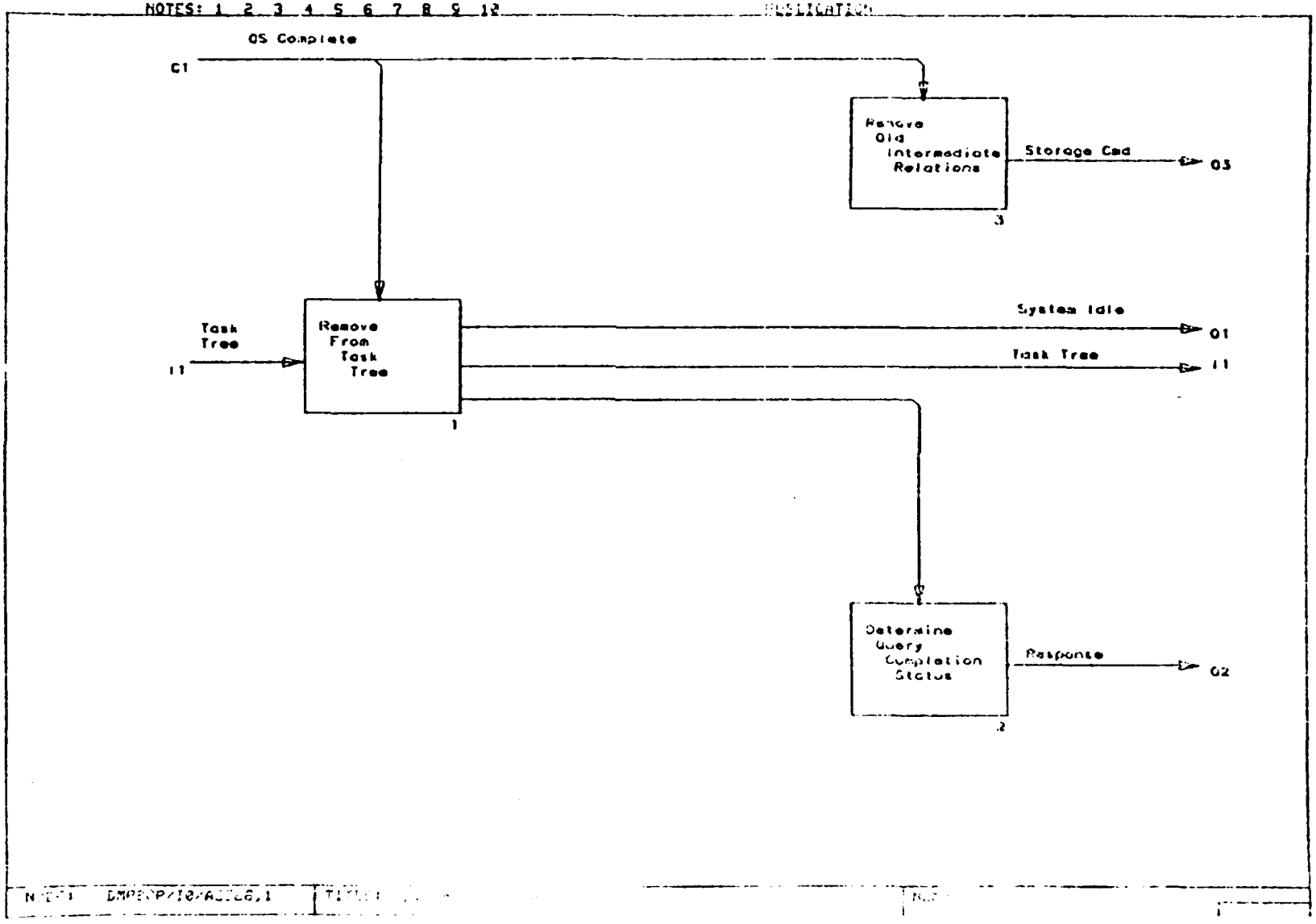
A22263 Remove Old Intermediate Relations causes the MSU to delete any old temporary relation(s) used by this query step.

AUTHOR: Capt Dale M. Pontiff
PROJECT: BACKEND
DATE: 11/27/84
REV:

WORKING DRAFT
X RECOMMENDED PUBLICATION

READER DATE CONTEXT:

NOTES: 1 2 3 4 5 6 7 8 9 12



A3 Shutdown

Abstract: Provides a safe, orderly method of terminating the operations of the DBMS.

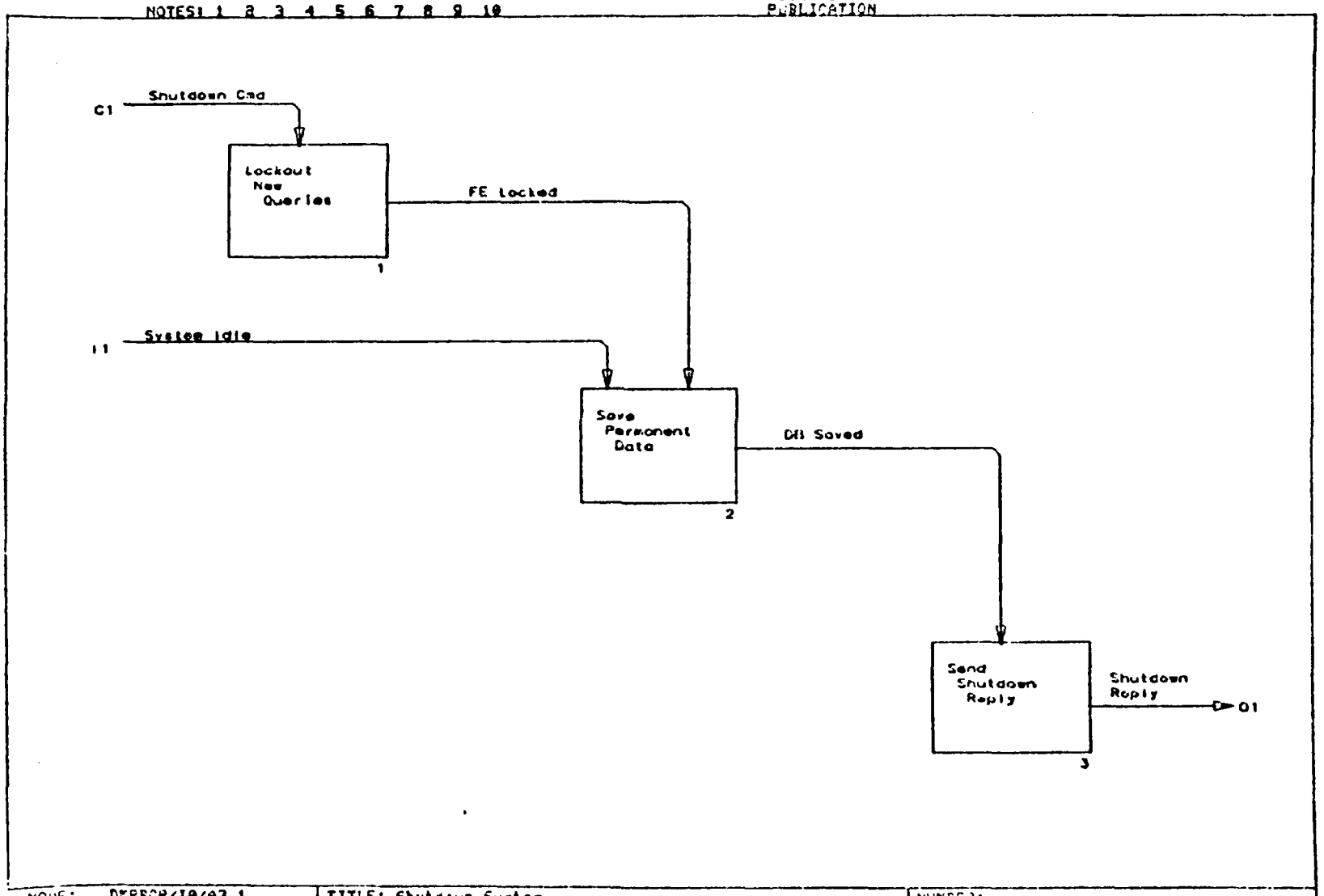
When the shutdown command is received, the FE is locked to prevent other queries from entering the system. Any existing queries in the system will run to completion (exception; stopped jobs will be killed). Once all queries have completed, all permanent data is saved, and a shutdown reply is sent.

A31 Lockout New Queries causes the frontend to stop listening for queries.

A32 Save Permanent Data causes any permanent files to be sent to the mass storage unit.

A33 Send Shutdown Reply Msgs informs the users that we are closed.

AUTHOR:Capt Dale M. Pontiff
PROJECT:BACKEND
DATE:11/27/84
REV:
WORKING
DRAFT
x RECOMMENDED
PUBLICATION
READER
DATE
CONTEXT:
NOTES: 1 2 3 4 5 6 7 8 9 10



NODE: DRPBCP/I0/A3.1
TITLE: Shutdown System
NUMBER:

Data Dictionary for the Date Elements

```
/*************************************************************/

NAME:              BCP Cmd
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A user command to modify the status of a
                   Query.
DATA TYPE:         BCP Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:           Cmd
                   Queue of Incoming BCP Msg
COMPOSITION:
ALIASES:
SOURCES:           Execute Preliminary  DBMS Functions (A212-2)
                   Determine BCP Action (A222-1)
DESTINATIONS:      Execute BCP Cmd (A222-2)
                   Queue FE Msgs (A212-5)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff


/*************************************************************/

NAME:              BCP Cmd Pkt
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       Part of a BCP command being passed from the FE
                   to the BCP.
DATA TYPE:         BCP Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:           Communication Packet between FE and BCP
COMPOSITION:
ALIASES:
SOURCES:           Provide Frontend DBMS Functions (A2-1)
                   Send FE Msgs (A21-3)
DESTINATIONS:      Provide BCP Functions (A2-2)
                   Receive BCP Msgs (A22-1)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:           BCP Init Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    An initialization packet sent from the FE to
                the BCP to determine if the BCP is active.
DATA TYPE:      BCP control command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between FE and BCP
COMPOSITION:
ALIASES:
SOURCES:        Init Frontend (A1-2)
DESTINATIONS:   Init BCP (A1-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/************************************************************/

NAME:           BCP Status Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control packet passed from the BCP to the
                FE to indicate that the BCP is active.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between FE and BCP
COMPOSITION:
ALIASES:
SOURCES:        Init BCP (A1-4)
DESTINATIONS:   Init Frontend (A1-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:           BCP Startup
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Startup control to power up and initialize the
                backend control processor.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Startup DBMS
COMPOSITION:
ALIASES:
SOURCES:        Human Intervention (A1 - C1)
DESTINATIONS:   Startup BCP (A1-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*************************************************************/

NAME:           Buffer Address
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Memory page address in the MBU.
DATA TYPE:      pointer
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Provide Mass Store Functions (A2-3)
                Provide QP DBMS Functions (A2-5)
DESTINATIONS:   Provide Memory Buffer Functions (A2-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A complete backend command send from host system.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:    DDL Cmd
                BCP Cmd
ALIASES:        Raw Cmd, Correct Cmd, Legal Cmd
SOURCES:
DESTINATIONS:
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Cmd Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a backend command being passed from
                host to backend.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Communication Packet between Host and Backend
                DDL Cmd
                BCP Cmd
ALIASES:
SOURCES:        Outside of Backend Environment (A-0)
DESTINATIONS:   Provide Relational DBMS Support (A-0)
                Provide DBMS Functions (A0-2)
                Provide Frontend DBMS Functions (A2-1)
                Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:           Correct Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A syntactically correct backend command
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Cmd
COMPOSITION:    DDL Cmd
                BCP Cmd
ALIASES:
SOURCES:        Analyse Syntax (A2122-1)
DESTINATIONS:   Verify Access (A2122-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/************************************************************/

NAME:           Correct Data
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A syntactically correct data file
DATA TYPE:      Input File
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Data
COMPOSITION:
ALIASES:
SOURCES:        Analyse Syntax (A2122-1)
DESTINATIONS:   Verify Access (A2122-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:           Correct Query
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A syntactically correct query string
DATA TYPE:      ASCII Query String
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Query
COMPOSITION:
ALIASES:
SOURCES:        Analyse Syntax (A2122-1)
DESTINATIONS:   Verify Access (A2122-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/************************************************************/

NAME:           Data
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Complete data input file send from host system.
DATA TYPE:      Input File
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:
ALIASES:        Raw Data, Correct Data, Legal Data
SOURCES:
DESTINATIONS:
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Data Dic
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Data Dictionary which defines all the domains
                and relations in the DB.
DATA TYPE:      Data Dictionary
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:
ALIASES:
SOURCES:        Manage DB Data Dic (A212-3)
                Determine FE Action (A212-1)
DESTINATIONS:   Manage DB Data Dic (A212-3)
                Execute Preliminary DBMS Functions (A212-2)
                Verify Access (A2122-2)
                Optimize Query (A2122-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Data Dic Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of the data dictionary being passed
                between MSU and the FE.
DATA TYPE:      Data Dictionary
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between MSU and the FE
COMPOSITION:
ALIASES:
SOURCES:        Provide Frontend Functions (A2-1)
                Provide Mass Storage Functions (A2-3)
DESTINATIONS:   Provide Frontend DBMS Functions (A2-1)
                Provide Mass Storage Functions (A2-3)
                Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Data Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a input data file (of new tuples)
                being passed from host to backend.
DATA TYPE:      Tuple data
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Communication Packet between Host and Backend
                     "          "       "    FE   and MSU
ALIASES:
SOURCES:        Outside of Backend Environment (A-0)
                Provide Frontend DBMS Function (A2-1)
                Provide Mass Store Functions (A2-3)
DESTINATIONS:   Provide Relational DBMS Support (A-0)
                Provide DBMS Functions (A0-2)
                Provide Frontend DBMS Functions (A2-1)
                Provide Mass Store Functions (A2-3)
                Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           DB Page Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a data page being passed between the
                MBU and a QP or the MSU.
DATA TYPE:      Relational Data Page
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between MBU and QP or
                MSU.
COMPOSITION:
ALIASES:
SOURCES:        Provide QP DBMS Functions (A2-5)
                Provide Memory Buffer Functions (A2-4)
                Provide Mass Storage Functions (A2-3)
DESTINATIONS:   Provide QP DBMS Functions (A2-5)
                Provide Memory Buffer Functions (A2-4)
                Provide Mass Storage Functions (A2-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           DB saved
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable indicating that the
                database has been saved on disk.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Save Permanent Data (A3-2)
DESTINATIONS:   Send Shutdown Reply (A3-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:            DDL Cmd
TYPE:            Data Element
PROJECT:         BCP
DESCRIPTION:     A DBA command to modify the Data Dictionary.
DATA TYPE:       Data Definition Language Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:         Cmd
COMPOSITION:
ALIASES:
SOURCES:         Execute Preliminary  DBMS Functions (A212-2)
                 Verify Access (A212-2)
DESTINATIONS:    Manage DB Data Dictionary (A212-3)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff

/*********************************************************/

NAME:            DDL Reply
TYPE:            Data Element
PROJECT:         BCP
DESCRIPTION:     Reply message for DDL commands.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:         Reply
COMPOSITION:
ALIASES:
SOURCES:         Manage DB Data Dic (A212-3)
DESTINATIONS:    Queue FE Msgs (A212-5)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           FE Locked
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable to indicate that the FE
                will not except new queries.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Lockout New Queries (A3-1)
DESTINATIONS:   Save Permanent Data (A3-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************/
NAME:           File Status
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Status of relation files to be accessed.
DATA TYPE:      File Status
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        BCP Table
DESTINATIONS:   Manage QP Assignment/Release (A222-4)
                Determine QP Assignment (A2224-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:              Free QP
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A control variable indicating that there is an
                   idle QP.
DATA TYPE:         Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Clear QP (A2225-3)
                   Manage Active Query Steps (A222-5)
DESTINATIONS:      Manage QP Assignment/Release (A222-4)
                   Select Highest Available Leaf (A2224-1)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff

/*********************************************************/

NAME:              Frontend Startup
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       Startup control to power up and initialize the
                   frontend.
DATA TYPE:         Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:           Startup DBMS
COMPOSITION:
ALIASES:
SOURCES:           Human Intervention (A1 - C1)
DESTINATIONS:      Startup Frontend (A1-1)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Init BCP
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable to boot the backend control
                processor.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Startup BCP (A1-3)
DESTINATIONS:   Init BCP (A1-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Init FE
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable to boot the frontend processor.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Startup Frontend (A1-1)
DESTINATIONS:   Init Frontend (A1-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:              Init QP
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A control variable to boot the query processors.
DATA TYPE:         Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Startup QP (A1-5)
DESTINATIONS:      Init QP (A1-6)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff


/*********************************************************/

NAME:              Leaf Count
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       Number of bottom-most leaves in the Task Tree.
DATA TYPE:         Integer
MIN VALUE:
MAX VALUE:
RANGE:             whole numbers
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Select Highest Available Leaf (A2224-1)
DESTINATIONS:      Determine QP Assignment (A2224-2)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Legal BCP Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A valid command to the BCP with proper user
                access.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Cmd
COMPOSITION:
ALIASES:
SOURCES:        Verfiy Access (A2122-2)
                Execute Preliminary DBMS Functions (A212-2)
DESTINATIONS:   Queue FE Msgs (A212-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Legal Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A valid backend command with proper user access.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Cmd
COMPOSITION:    DDL Cmd
                BCP Cmd
ALIASES:
SOURCES:        Verfiy Access (A2122-2)
DESTINATIONS:   Log Transaction (A2122-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Legal Data
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A valid input data file with proper user access.
DATA TYPE:      Input File
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Data
                Queue of Outgoing FE Msgs
COMPOSITION:
ALIASES:
SOURCES:        Verfiy Access (A2122-2)
                Execute Preliminary DBMS Functions (A212-2)
DESTINATIONS:   Queue FE Msgs (A212-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Legal DDL Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A valid Data Definition Command with proper user
                access.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Cmd
COMPOSITION:
ALIASES:
SOURCES:        Verfiy Access (A2122-2)
                Execute Preliminary DBMS Functions (A212-2)
DESTINATIONS:   Manage DB Data Dic (A212-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Legal Query
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A valid Query with proper user access.
DATA TYPE:      ASCII Query String
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Query
COMPOSITION:
ALIASES:
SOURCES:        Verfiy Access (A2122-2)
DESTINATIONS:   Log Transaction (A2122-3)
                Optimize Query (A2122-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*************************************************************/

NAME:           Load Init Pages
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Structure tells the paging system what types
                of pages are need during the initial load and
                where to load the pages in the MBU.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Allocate Buffer Space (A2225-1)
DESTINATIONS:   Manage Paging (A2225-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Multiple QPs
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable used to determine if more
                than one QP has been assigned to a Query Step.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Determine QP Assignment (A2224-2)
DESTINATIONS:   Create Compression Node (A2224-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Optimized Query
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A Query stored in an optimized query tree.
DATA TYPE:      Query Tree
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Outgoing FE Msgs
COMPOSITION:
ALIASES:        Query Tree
SOURCES:        Optimized Query (A2122-4)
                Execute Preliminary DBMS Functions (A212-2)
                Determine BCP Action (A222-1)
DESTINATIONS:   Queue FE Msgs (A212-5)
                Add to Task Tree (A222-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Optimized Query Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of an optimized query tree being passed
                from the FE to the BCP.
DATA TYPE:      Comm Packet
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between FE and BCP
COMPOSITION:
ALIASES:
SOURCES:        Provide Frontend DBMS Functions (A2-1)
                Send FE Msgs (A21-3)
DESTINATIONS:   Provide BCP Functions (A2-2)
                Receive BCP Msgs (A22-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           Output
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A complete output relations for a retrieval
                Query from the MSU.
DATA TYPE:      Output Relation (file)
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
                Queue of Outgoing BCP Msgs
COMPOSITION:
ALIASES:
SOURCES:        Determine FE Action (A212-1)
                Build Reply (A212-4)
DESTINATIONS:   Queue FE Msgs (A212-5)
                Build Reply (A212-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:              Output Pkt
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       Part of a output relation being passed between
                   processors.
DATA TYPE:         Output Relation
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:       Communication Packet between Host and Backend
ALIASES:
SOURCES:           Provide Relational DBMS Support (A-0)
                   Provide DBMS Functions (A0-2)
                   Provide Frontend DBMS Functions (A2-1)
                   Provide Mass Store Functions (A2-3)
                   Send FE Msgs (A21-3)
DESTINATIONS:      Provide Frontend DBMS Functions (A2-1)
                   Receive FE Msgs (A21-1)
                   Outside of Backend Environment (A-0)
RELATED REQUIREMENT NUMBER:

VERSION:            1.0
DATE:               11/16/84
AUTHOR:             Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:           Page Request
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A page request from a QP.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Queue of Incoming BCP Msgs
ALIASES:
SOURCES:        Determine BCP Action (A222-1)
DESTINATIONS:   Manage Active Query Step (A222-5)
                Manage Paging (A2225-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/************************************************************/

NAME:           Page Request Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a page request message being passed
                from a QP to the BCP.
DATA TYPE:      Paging control data
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between BCP and QP
COMPOSITION:
ALIASES:
SOURCES:        Provide QP DBMS Functions (A2-5)
DESTINATIONS:   Provide BCP Functions (A2-2)
                Receive BCP Msgs (A22-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Paging Info
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Structure used to tell the QPs which pages of
                the MBU to access and what is stored in each
                page.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Manage Active Query Steps (A222-5)
                Manage Paging (A2225-2)
DESTINATIONS:   Queue BCP Msgs (A222-7)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Paging Info Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a paging message being passed from the
                BCP to a QP.
DATA TYPE:      Paging control data
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between BCP and QP
COMPOSITION:
ALIASES:
SOURCES:        Provide BCP Functions (A2-2)
                Send BCP Msgs (A22-3)
DESTINATIONS:   Provide QP DBMS Functions (A2-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Preempt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable to cause a Query to be
                preempted.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Execute Cmd (A222-2)
DESTINATIONS:   Determine Which Query Steps To Preempt
                     (A2224-4)
                Manage QP Assignment/Release (A222-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Preempt QS
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable to cause a Query Step to be
                preempted (stop any new paging).
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Determine Which Query Steps To Preemmpt
                          (A2224-4)
                Manage QP Assignment/Release (A222-4)
DESTINATIONS:   Manage Active Query Steps (A222-5)
                Manage Paging (A2225-2)
                Clear QP (A2225-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Priority
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A priority value to determine the order in
                which query steps are executed.
DATA TYPE:      Integer
MIN VALUE:      0 (high priority)
MAX VALUE:      255 (low priority)
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Determine Priority (A2223-1)
DESTINATIONS:   Join to Task Tree (A2223-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*********************************************************/

NAME:           QP Init Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    An initialization packet sent from the BCP to
                the QPs to determine which QPs are active.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between BCP and QP
COMPOSITION:
ALIASES:
SOURCES:        Init BCP (A1-4)
DESTINATIONS:   Init QP (A1-6)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           QP Startup
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Startup control to power up and initialize the
                query processors.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Startup DBMS
COMPOSITION:
ALIASES:
SOURCES:        Human Intervention (A1 - C1)
DESTINATIONS:   Startup QP (A1-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           QP Status Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control packet passed from the QPs to the
                BCP to indicate which QPs are active.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between BCP and QP
COMPOSITION:
ALIASES:
SOURCES:        Init QP (A1-6)
DESTINATIONS:   Init BCP (A1-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           QP Stopped
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable indicating the a QP has
                stopped execution of a query step.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Clear QP (A2225-3)
                Manage Active Query Steps (A222-5)
DESTINATIONS:   Manage QP Assignment/Release (A222-4)
                Trim Branch (A2224-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           QS Complete
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A control variable indicating that a
                Query Step has completed.
DATA TYPE:      Control Variable
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:        Manage Paging (A2225-2)
                Manage Active Query Steps (A222-5)
DESTINATIONS:   Update Task Tree (A222-6)
                Clear QP (A2225-3)
                Remove Old Intermediate Relations (A2226-3)
                Remove from Task Tree (A2226-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Query
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Complete ASCII query string send from host system.
DATA TYPE:      Query String
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:
ALIASES:        Raw Query, Correct Query, Legal Query
SOURCES:
DESTINATIONS:
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff


/*************************************************************/

NAME:           Query Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a Query being passed from the
                Host System to the Backend System
DATA TYPE:      ASCII String
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between Host and Backend
COMPOSITION:
ALIASES:
SOURCES:        Outside of Backend Environment (A-0)
DESTINATIONS:   Provide Relational DBMS Support (A-0)
                Provide DBMS Functions (A0-2)
                Provide Frontend DBMS Functions (A2-1)
                Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:              Query Status
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       Status of a Query Step
DATA TYPE:         Query Status
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Manage QP Assignment/Release (A222-4)
                   Manage Active Query Steps (A222-5)
                   Determine QP Assignment (A2224-2)
                   Manage Paging (A2225-2)
DESTINATIONS:      Manage QP Assignment/Release (A222-4)
                   Manage Active Query Steps (A222-5)
                   Determine QP Assignment (A2224-2)
                   Manage Paging (A2225-2)
                   Determine Which Query Steps to Preempt
                        (A2224-4)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:              Query Step
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A single relational operations to be performed
                   on one or two relational files.
DATA TYPE:         Query Step
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Select Highest Available Leaf (A2224-1)
                   Manage QP Assignment/Release (A222-4)
                   Manage Active Query Steps (A222-5)
                   Allocate Buffer Space (A2225-1)
                   Determine QP Assignment (A2224-3)
DESTINATIONS:      Queue BCP Msgs (A222-7)
                   Manage Active Query Steps (A222-5)
                   Allocate Buffer Space (A2225-1)
                   Determine QP Assignment (A2224-3)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Query Step Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a query step being passed from the BCP
                to a QP.
DATA TYPE:      Query Step (operation)
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between BCP and QP
COMPOSITION:
ALIASES:
SOURCES:        Provide BCP Functions (A2-2)
                Send BCP Msgs (A22-3)
DESTINATIONS:   Provide QP DBMS Functions (A2-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Query Tree
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    An Optimized Query rebuilt within the BCP.
DATA TYPE:      Query Tree
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Task Tree
COMPOSITION:    Query Header
                Query Step
ALIASES:        Optimized Query
SOURCES:        Execute Cmd (A222-2)
                Determine BCP Action (A222-1)
DESTINATIONS:   Add to Task Tree (A222-3)
                Determine Priority (A2223-1)
                Join to Task Tree (A2223-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************/

NAME:              Queue of Incoming BCP Msgs
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A queue of message received by the BCP for
                   processing.
DATA TYPE:         Queue
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:       Optimized Query
                   BCP Cmd
                   Page Request
ALIASES:
SOURCES:           Receive BCP Msgs (A22-1)
DESTINATIONS:      Execute BCP BDMS Functions (A22-2)
                   Determine BCP Action (A222-1)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:              Queue of Incoming FE Msgs
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A queue of message received by the FE for
                   processing.
DATA TYPE:         Queue
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:       Query
                   Cmd
                   Data
                   Response
                   Data Dic
                   Output
ALIASES:
SOURCES:           Receive FE Msgs (A21-1)
DESTINATIONS:      Execute FE DBMS Functions (A21-2)
                   Determine FE Action (A212-1)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:              Queue of Outgoing BCP Msgs
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A queue of message the BCP must send to other
                   processors (FE, MSU, QP).
DATA TYPE:         Queue
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:       Response
                   Storage Cmd
                   Query Step
                   Paging Info
ALIASES:
SOURCES:           Queue BCP Msgs (A222-7)
                   Execute BCP DBMS Functions (A22-2)
DESTINATIONS:      Send BCP Msgs (A22-3)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:              Queue of Outgoing FE Msgs
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A queue of message the FE must send to other
                   processors (host, BCP, MSU).
DATA TYPE:         Queue
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:       Optimized Query
                   BCP Cmd
                   Valid Data
                   Storage Cmd
                   Reply
                   Output
ALIASES:
SOURCES:           Queue FE Msgs (A212-5)
                   Execute FE DBMS Function (A21-2)
DESTINATIONS:      Send FE Msgs (A21-3)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Raw Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A complete backend command sent from host system.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:    DDL Cmd
                BCP Cmd
ALIASES:
SOURCES:        Receive FE Msgs (A21-1)
DESTINATIONS:   Execute FE DBMS Functions (A21-2)
                Execute Preliminary DBMS Functions (A212-2)
                Analyze Syntax (A2122-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Raw Data
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Complete data input file sent from host system.
DATA TYPE:      Input File
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:
ALIASES:
SOURCES:        Receive FE Msgs (A21-1)
DESTINATIONS:   Execute FE DBMS Functions (A21-2)
                Execute Preliminary DBMS Functions (A212-2)
                Analyze Syntax (A2122-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Raw Query
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Complete ASCII query string sent from host system.
DATA TYPE:      Query String
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
COMPOSITION:
ALIASES:
SOURCES:        Receive FE Msgs (A21-1)
DESTINATIONS:   Execute FE DBMS Functions (A21-2)
                Execute Preliminary DBMS Functions (A212-2)
                Analyze Syntax (A2122-2)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Reply
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A reply message from the backend to the host.
DATA TYPE:      Reply
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Outgoing FE Msgs
COMPOSITION:    DDL Reply
                Shutdown Reply
ALIASES:
SOURCES:        Build Reply (A212-4)
DESTINATIONS:   Queue FE Msgs (A212-5)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Reply Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a output reply being passed from
                backend to host system.
DATA TYPE:      Reply to Backend Command or Update Query
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Communication Packet between Host and Backend
ALIASES:
SOURCES:        Provide Relational DBMS Support (A0)
                Provide DBMS Functions (A0-2)
                Provide Frontend DBMS Functions (A2-1)
                Send FE Msgs (A21-3)
DESTINATIONS:   Outside of Backend Environment (A0)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff

/*********************************************************/

NAME:           Resource Status
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Status of QPs.
DATA TYPE:      QP Status
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:
DESTINATIONS:   Determine QP Assignment (A2224-2)
                Manage QP Assignment/Release (A222-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Response
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A complete response message concerning a Query
                or BCP Cmd from the BCP.
DATA TYPE:      Response
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Queue of Incoming FE Msgs
                Queue of Outgoing BCP Msgs
COMPOSITION:
ALIASES:
SOURCES:        Determine Query Completion Status (A2226-2)
                Update Task Tree (A222-6)
                Determine FE Action (A212-1)
DESTINATIONS:   Queue BCP Msg (A222-7)
                Build Reply (A212-4)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff

/*********************************************************/

NAME:           Response Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a response passed between the BCP and
                FE.
DATA TYPE:      Response
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Communication Packet between BCP and FE.
ALIASES:
SOURCES:        Provide BCP Functions (A2-2)
                Send BCP Msgs (A22-3)
DESTINATIONS:   Provide FE DBMS Functions (A2-1)
                Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:           Shutdown Cmd
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A command from the host telling the backend to
                begin shutting down.
DATA TYPE:      Backend Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Cmd
COMPOSITION:
ALIASES:
SOURCES:        Outside of Backend Environment (A0 - I1)
DESTINATIONS:   Shutdown System (A0-3)
                Lockout New Queries (A3-1)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff

/*********************************************************/


NAME:           Shutdown Reply
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A reply from the backend informing the host
                that the backend is idle after receiving a
                shutdown command.
DATA TYPE:      Reply
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Reply
COMPOSITION:
ALIASES:
SOURCES:        Shutdown System (A0-3)
                Send Shutdown Reply (A3-3)
DESTINATIONS:   Outside of Backend Environment (A0 - O1)
RELATED REQUIREMENT NUMBER:

VERSION:         1.0
DATE:            11/16/84
AUTHOR:          Capt Dale M. Pontiff

/*********************************************************/
```

NAME: Startup DBMS
TYPE: Data Element
PROJECT: BCP
DESCRIPTION: Startup the backend relational DBMS. (Manual)
DATA TYPE: Human
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION: Frontend Startup
BCP Startup
QP Startup
ALIASES:
SOURCES: Outside of Backend Environment (A0)
DESTINATIONS: Provide Relational DBMS Support (A0)
Initialize Database System (A0-1)
RELATED REQUIREMENT NUMBER:

VERSION: 1.0
DATE: 11/16/84
AUTHOR: Capt Dale M. Pontiff

```
/*********************************************************/

NAME:              Storage Cmd
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A command to the MSU to create, delete or
                   access a data file.
DATA TYPE:         MSU Cmd
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:           Queue of Outgoing FE Msgs
COMPOSITION:
ALIASES:
SOURCES:           Verify Access (A2122-2)
                   Execute Preliminary DBMS Functions (A212-2)
                   Manage DB Data Dic (A212-3)
                   Build Reply (A212-4)
                   Remove Old Intermediate Relations (A2226-3)
                   Update Task Tree (A222-6)
                   Manage Paging (A2225-3)
                   Manage Active Query Steps (A222-5)
DESTINATIONS:      Queue FE Msgs (A212-5)
                   Queue BCP Msgs (A222-7)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Storage Cmd Pkt
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    Part of a MSU command being passed from the
                FE or BCP to the MSU.
DATA TYPE:      Mass Storage Command
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:        Communication Packet between the FE or BCP
                to the MSU
COMPOSITION:
ALIASES:
SOURCES:        Provide Frontend DBMS Functions (A2-1)
                Send FE Msgs (A21-3)
                Provide BCP Functions (A2-2)
                Send BCP Msgs (A22-3)
DESTINATIONS:   Provide Mass Store Functions (A2-3)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*********************************************************/

NAME:              System Idle
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A control variable indicating that the backend
                   is currently idle.
DATA TYPE:         Control Flag
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Provide DBMS Functions (A0-2)
                   Provide BCP Functions (A2-2)
                   Execute BCP DBMS Functions (A22-2)
                   Update Task Tree (A222-6)
                   Remove from Task Tree (A2226-1)
DESTINATIONS:      Shutdown System (A0-3)
                   Save Permanent Data (A3-2)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:              System Ready
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A control variable indicating that the backend
                   has been initialized.
DATA TYPE:         Control Flag
MIN VALUE:
MAX VALUE:
RANGE:             On --> System Ready;
                   Off --> System is inoperative
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Initialize Database System (A0-1)
                   Init Frontend (A1-2)
DESTINATIONS:      Provide DBMS Functions (A0-2)
                   Provide Frontend DBMS Functions (A2-1)
                   Receive FE Msgs (A21-1)
RELATED REQUIREMENT NUMBER:

VERSION:            1.0
DATE:               11/16/84
AUTHOR:             Capt Dale M. Pontiff
```

```
/************************************************************/

NAME:           Task Tree
TYPE:           Data Element
PROJECT:        BCP
DESCRIPTION:    A colection of active Query Trees within the BCP.
DATA TYPE:      Task Tree
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:    Query Tree
ALIASES:
SOURCES:        Add to Task Tree (A222-3)
                Update Task Tree (A222-6)
                Remove from Task Tree (A2226-1)
                Trim Branch (A2224-3)
                Create Compression Node (A2224-5)
                Manage QP Assignment/Release (A222-4)
                Join to Task Tree (A2223-2)
DESTINATIONS:   Add to Task Tree (A222-3)
                Update Task Tree (A222-6)
                Remove from Task Tree (A2226-1)
                Trim Branch (A2224-3)
                Create Compression Node (A2224-5)
                Manage QP Assignment/Release (A222-4)
                Join to Task Tree (A2223-2)
                Select Highest Available Leaf (A2224-1)
RELATED REQUIREMENT NUMBER:

VERSION:        1.0
DATE:           11/16/84
AUTHOR:         Capt Dale M. Pontiff
```

```
/*************************************************************/

NAME:              Trimmed Branch
TYPE:              Data Element
PROJECT:           BCP
DESCRIPTION:       A Query Tree which has been removed from the
                   Task Tree.
DATA TYPE:         Query Tree
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
ALIASES:
SOURCES:           Manage QP Assignment/Release (A222-4)
                   Trim Branch (A2224-5)
DESTINATIONS:      Execute Cmd (A222-2)
RELATED REQUIREMENT NUMBER:

VERSION:           1.0
DATE:              11/16/84
AUTHOR:            Capt Dale M. Pontiff
```

Data Dictionary for the Activity Boxes

```
/*************************************************************/

NAME:           Add to Task Tree
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2223
DESCRIPTION:    Phase the new query into the task tree
                according to its priority.
INPUTS:         O1, Task Tree
OUTPUTS:        O1, Task Tree
CONTROLS:       C1, Query Tree (Optimized Query)
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Allocate Buffer Space
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22251
DESCRIPTION:    Determines how much buffer spaces is needed
                (and available) by the Query Step.  It then
                forwards the Query Step down to the QP, and
                causes the paging system to load the initial
                pages of the relation.
INPUTS:
OUTPUTS:        O1, Query Step
                O2, Load Init Pages
CONTROLS:       C1, Query Step
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2225, Manage Active Query Steps
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Analyze Syntax
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A21221
DESCRIPTION:    Checks syntax for all commands and queries.
                It also verifies if the relations and/or
                fields requested exist in the database.
INPUTS:
OUTPUTS:        O1, Correct Query, Data or Cmd
CONTROLS:       C1, Raw Query, Data, or Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2122, Execute Preliminary DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Build Reply
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2124
DESCRIPTION:    Receives responses from the BCP and Output
                relations from the MSU and formats the data to
                be forwarded to the host system.
INPUTS:
OUTPUTS:        O1, Output, Reply and/or Storage Cmd
CONTROLS:       C1, Response or Output
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A212, Execute FE DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Clear QP
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22253
DESCRIPTION:    Cleans up the QP and prepares it to receive a
                new Query Step.
INPUTS:
OUTPUTS:        O1, Free QP
                O2, QP Stopped
CONTROLS:       C1, Preempt QS
                C2, QS Complete
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2225, Manage Active Query Steps
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Create Compression Node
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22243
DESCRIPTION:    Adds a compression node above a node that was
                split between two or more QPs.
INPUTS:         I1, Task Tree
OUTPUTS:        O1, Task Tree
CONTROLS:       C1, Multiple QPs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2224, Manage QP Assignment/Release

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Determine BCP Actions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2221
DESCRIPTION:    Reads the message at the top of the queue and
                takes the appropriate action.
INPUTS:
OUTPUTS:        O1, BCP Cmd
                O2, Optimized Query (Query Tree)
                O3, Page Request
CONTROLS:       C1, Queue of Incoming BCP Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Determine FE Actions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2121
DESCRIPTION:    Reads the top message in the queue and calls
                the correct module for that message type.
INPUTS:
OUTPUTS:        O1, Raw Query, Data or Cmd
                O2, Data Dic
                O3, Response or Output
CONTROLS:       C1, Queue of Incoming FE Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A212, Execute FE DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Determine Priority
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22231
DESCRIPTION:    Generates a priority rating for a query.
INPUTS:
OUTPUTS:        O1, Priority
CONTROLS:       C1, Query Tree
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2223, Add to Task Tree

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Determine QS Requirements
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22242
DESCRIPTION:    Decides if a QP should be allocated to work on
                this query step based on system status.
INPUTS:         I1, Query Status
                I2, File Status
                I3, Resource Status
OUTPUTS:        O1, Query Step
                O2, Query Status
                O3, Multiple QPs
CONTROLS:       C1, Leaf Count
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2224, Manage QP Assignment/Release

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

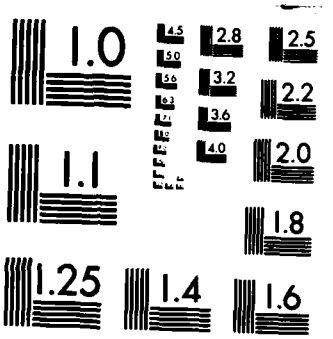
```
/*************************************************************/

NAME:           Determine Query Completion Status
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22262
DESCRIPTION:    Decides if the query is completed.  If so, it
                tells the frontend the location of any answer
                relation.
INPUTS:
OUTPUTS:        O1, Response
CONTROLS:       C1, Check Completion
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2226, Update Task Tree
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Determine Which Query Step to Preempt
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22244
DESCRIPTION:    Used to stop/abort a job.  It causes any QPs
                working on the terminated job to stop after
                the completion of the current input page.
INPUTS:         I1, Query Status
OUTPUTS:        O1, Preempt QS
CONTROLS:       C1, Preempt
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2224, Manage QP Assignment/Release

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Execute BCP Cmd
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2222
DESCRIPTION:    Allows some external job control commands to
                affect the system job scheduler.
INPUTS:
OUTPUTS:        O1, Response
                O2, Preempt
                O3, Query Tree
CONTROLS:       C1, BCP Cmd
                O2, Trimmed Branch
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Execute BCP DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A222
DESCRIPTION:    Executes BCP commands, schedules query steps
                to the QPs, and manages the paging algorithm
                of each operation.
INPUTS:
OUTPUTS:        O1, Queue of Outgoing BCP Msgs
CONTROLS:       C1, System Idle
                C2, Queue of Incoming BCP Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A22, Provide BCP Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Execute FE DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A212
DESCRIPTION:    Acts on incoming messages from the other
                processors.  Its major functions are; to
                receive and validate queries/commands from the
                host, manage the Data Dictionary, pass the
                queries down to the BCP, and send replies back
                to the host.
INPUTS:
OUTPUTS:        O1, Queue of Outgoing FE Msgs
CONTROLS:       C1, Queue of Incoming FE Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A21, Provide Frontend DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Execute Preliminary DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2122
DESCRIPTION:    Receives raw input from the host system.  It
                checks the syntax and user access rights, and
                logs the transactions.  If it is a retrieval
                query, it optimizes the query tree.
INPUTS:         I1, Data Dic
OUTPUTS:        O1, Optimized Query, Legal Data,
                      Legal BCP Cmd and/or Storage Cmd
                O2, Legal DDL Cmd
CONTROLS:       C1, Raw Query, Data or Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A212, Execute FE DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

AD-A151 892 BACKEND CONTROL PROCESSOR FOR A MULTI-PROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 3/3

UNCLASSIFIED D M PONTIFF DEC 84 AFIT/GCS/ENG/84D-22 F/G 9/2 NL



```
/*****************************************************************/

NAME:           Init BCP
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A14
DESCRIPTION:    Causes the BCP to initialize system tables and
                verify that at least one QP is available for
                use.  Sends a message to the FE after
                initialization is complete.
INPUTS:         I1, BCP Init Pkt
                O1, QP Status Pkt
OUTPUTS:        O1, QP Init Pkt
                I1, BCP Status Pkt
CONTROLS:       C1, Init BCP
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Init Frontend
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A12
DESCRIPTION:    Causes the FE to initialize system tables and
                verify that the BCP is available for use.
INPUTS:         O1, BCP Status Pkt
OUTPUTS:        O1, BCP Init Pkt
                O2, System Ready
CONTROLS:       C1, Init FE
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Init QP
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A16
DESCRIPTION:    Causes the QP to initialize any internal
                fields or tables.  Sends a message to the BCP
                upon completion.
INPUTS:         I1, QP Init Pkt
OUTPUTS:        I1, QP Status Pkt
CONTROLS:       C1, Init QP
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff



/*****************************************************************/

NAME:           Initialize Database System
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A1
DESCRIPTION:    Entails all necessary steps needed to bring
                the DBMS up as a functioning unit (i.e.
                supply power, load OS, initialize system
                tables, etc.).
INPUTS:
OUTPUTS:        O1, System Ready
CONTROLS:       C1, Startup DBMS
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A0, Provide Relational DBMS Support

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Join to Task Tree
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22232
DESCRIPTION:    Adds the incoming query tree into the task
                tree as another branch.  The query tree will
                be joined at the root node based on its
                priority.
INPUTS:         I1, Priority
                I2, Task Tree
OUTPUTS:        O1, Task Tree
CONTROLS:       C1, Query Tree
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2223, Add to Task Tree

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Lockout New Queries
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A3
DESCRIPTION:    Causes the frontend to stop listening for
                queries.
INPUTS:
OUTPUTS:        O1, FE Locked
CONTROLS:       C1, Shutdown Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A3, Shutdown System
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Log Transaction
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A21223
DESCRIPTION:    Provides a transaction log of all queries and
                commands (both for backup and security
                purposes).
INPUTS:
OUTPUTS:
CONTROLS:       C1, Legal Query, or Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2122, Execute Preliminary DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Manage Active Query Step
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2225
DESCRIPTION:    Directs the QPs and controls the system paging
                of the MBU and MSU.
INPUTS:         I1, Query Status
                I2, Page Request
OUTPUTS:        O1, Query Step or Paging Info
                O2, Storage Cmd
                O3, QS Complete
                I1, Query Status
                C1, Free QP
                C2, QP Stopped
CONTROLS:       C1, Preempt QS
                C2, Query Step
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Manage DB Data Dic
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2123
DESCRIPTION:    Responsible for maintaining the database data
                dictionary.  If the input message was a valid
                DDL command, it modifies the Data Dictionary
                as needed.
INPUTS:         I1, Data Dic
OUTPUTS:        O1, Data Dic, DDL Reply, or Storage Cmd
CONTROLS:       C1, Legal DDL Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A212, Execute FE DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Manage Paging
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22252
DESCRIPTION:    Handles the system paging algorithms.
INPUTS:         I1, Query Status
OUTPUTS:        O1, Paging Info
                O2, Query Status
                O3, Storage Cmd
                O4, QS Complete
CONTROLS:       C1, Preempt QS
                C2, Page Request
                C3, Load Init Pages
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2225, Manage Active Query Steps
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Manage QP Assignment/Release
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2224
DESCRIPTION:    Selects the next Query Step to be executed,
                and determines which Query Steps must be
                preempted when a job is stopped.
INPUTS:         I1, Task Tree
                I2, File Status
                I3, Resource Status
                O3, Query Status
OUTPUTS:        O1, Query Step
                O2, Preempt QS
                O3, Query Status
                I1, Task Tree
                C1, Trimmed Branch
CONTROLS:       C1, Preempt
                O1, Free QP
                O2, QP Stopped
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Optimize Query
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A21224
DESCRIPTION:    Arranges a complex query into a relatively
                efficient query form (tree).
INPUTS:         I1, Data Dic
OUTPUTS:        O1, Optimized Query
CONTROLS:       C1, Legal Query
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2122, Execute Preliminary DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Provide BCP Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22
DESCRIPTION:    Responsible for scheduling query tasks and
                managing the system paging.
INPUTS:
OUTPUTS:        O1, System Idle
                O2, Query Step or Paging Info Pkt
                O3, Storage Cmd Pkt
                C1, Response Pkt
CONTROLS:       C1, Optimized Query or BCP Cmd Pkt
                O2, Page Request Pkt
MECHANISMS:     M1, Backend Control Processor
ALIASES:
PARENT ACTIVITY: A2, Provide DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Provide DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2
DESCRIPTION:    Contains the necessary DBMS functions needed
                to allow queries and commands to be levied
                against the existing DB.
INPUTS:         I1, Query, Data, or Cmd Pkt
OUTPUTS:        O1, Output or Reply Pkt
                O2, System Idle
CONTROLS:       C1, System Ready
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A0, Provide Relational DBMS Support

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Provide Frontend DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A21
DESCRIPTION:    Responsible for communications between the
                backend system and the outside world.  It is
                also responsible for most of the database
                management functions not directly related to
                relational operations against the database
                (such as syntax checks, security checks,
                transaction log, and query optimization).
INPUTS:         I1, Query, Data, or Cmd Pkt
                O4, Data Dic or Output Pkt
OUTPUTS:        O1, Output or Reply Pkt
                O2, Optimized Query or BCP Cmd Pkt
                O3, Storage Cmd Pkt
                O4, Data or Data Dic Pkt
CONTROLS:       C1, System Ready
                O2, Response Pkt
MECHANISMS:     M1, Frontend
ALIASES:
PARENT ACTIVITY: A2, Provide DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Provide Mass Store Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A23
DESCRIPTION:    Responsible for file management.  It provides
                permanent storage of the existing database,
                plus temporary storage of any intermediate
                relations created during a query.
INPUTS:         I1, Data or Data Dic Pkt
                O2, DB Page Pkt
OUTPUTS:        O1, Buffer Address
                O2, DB Page Pkt
CONTROLS:       C1, Storage Cmd Pkt
MECHANISMS:     M1, Mass Store Unit
ALIASES:
PARENT ACTIVITY: A2, Provide DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Provide Memory Buffer Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A24
DESCRIPTION:    Provides very fast scratch pad memory for the
                QPs to manipulated data.
INPUTS:         I1, DB Page Pkt
                O1, DB Page Pkt
OUTPUTS:        O1, DB Page Pkt
                I1, DB Page Pkt
CONTROLS:       C1, Buffer Address
MECHANISMS:     M1, Memory Buffer Unit
ALIASES:
PARENT ACTIVITY: A2, Provide DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Provide QP DBMS Functions
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A25
DESCRIPTION:    Provides the relational operations (select,
                project, join, product, union, difference,
                intersection), update operations (insert,
                delete, modify), and miscellaneous operations
                (min, max, count, sort, sum) that actual act
                on the data within the DBMS.
INPUTS:         I1, DB Page Pkt
OUTPUTS:        O1, Buffer Address
                I1, DB Page Pkt
                C1, Page Request Pkt
CONTROLS:       C1, Query Step or Paging Info Pkt
MECHANISMS:     M1, Query Processor
ALIASES:
PARENT ACTIVITY: A2, Provide DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Provide Relation DBMS Support
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A-0
DESCRIPTION:    A relational database management system.
INPUTS:         I1, Query, Data, or Cmd Pkt
OUTPUTS:        O1, Output or Reply Pkt
CONTROLS:       C1, Startup DBMS
MECHANISMS:     M1, Backend DBMS
ALIASES:
PARENT ACTIVITY:

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Queue BCP Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2227
DESCRIPTION:    Places any outgoing communication messages in
                a queue.
INPUTS:
OUTPUTS:        O1, Queue of Outgoing BCP Msgs
CONTROLS:       C1, Storage Cmd
                C2, Query Step or Paging Info
                C3, Response
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*************************************************************/

NAME:           Queue FE Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2125
DESCRIPTION:    Places any outgoing communication messages in
                a queue.
INPUTS:
OUTPUTS:        O1, Queue of Outgoing FE Msgs
CONTROLS:       C1, Output, Reply and/or Storage Cmd
                C2, Data Dic, DDL Reply, or Storage Cmd
                C3, Optimized Query, Legal Data,
                     Legal BCP Cmd, and/or Storage Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A212, Execute FE DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Receive BCP Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A221
DESCRIPTION:    Listens for incoming messages from the
                frontend or the query processors.
INPUTS:
OUTPUTS:        O1, Queue of Incoming BCP Msgs
CONTROLS:       C1, Optimized Query or BCP Cmd Pkt
                C2, Page Request Pkt
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A22, Provide BCP Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff

/*************************************************************/

NAME:           Receive FE Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A211
DESCRIPTION:    Listens for incoming messages from the host
                system or from other components of the Backend
                System.  The messages are converted into a
                useable form for the FE.
INPUTS:
OUTPUTS:        O1, Queue of Incoming FE Msgs
CONTROLS:       C1, Data Dic or Output Pkt
                C2, Response Pkt
                C3, Query, Data, or Cmd Pkt
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A21, Provide Frontend DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Remove From Task Tree
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22261
DESCRIPTION:    Removes the QS from the Task Tree.
INPUTS:         I1, Task Tree
OUTPUTS:        O1, System Idle
                O2, Task Tree
                O3, Check Completion
CONTROLS:       C1, QS Complete
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2226, Update Task Tree
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Remove Old Intermediate Relationss
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22263
DESCRIPTION:    Causes the MSU to delete any old temporary
                relation(s) used by this query step.
INPUTS:
OUTPUTS:        O1, Storage Cmd
CONTROLS:       C1, QS Complete
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2226, Update Task Tree
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Save Permanent Data
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A32
DESCRIPTION:    Causes the frontend to stop listening for
                queries.
INPUTS:
OUTPUTS:        O1, DB Saved
CONTROLS:       C1, System Idle
                C2, FE Locked
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A3, Shutdown System
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Select Highest Available Leaf
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22241
DESCRIPTION:    Chooses the leaf node with the highest
                priority.  If it is directed to reselect, it
                ignores all previously selected leaves.
INPUTS:         I1, Task Tree
OUTPUTS:        O1, Leaf Count
CONTROLS:       C1, Free QP
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2224, Manage QP Assignment/Release

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*************************************************************/

NAME:           Send BCP Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A223
DESCRIPTION:    Converts the BCP's internal data structures
                into a form that can be transferred to the
                other processors.  It sends responses to the
                FE, storage commands to the MSU, and query
                steps and paging information to the QPs.
INPUTS:
OUTPUTS:        O1, Response Pkt
                O2, Storage Cmd Pkt
                O3, Query Step or Paging Info Pkt
CONTROLS:       C1, Queue of Outgoing BCP Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A22, Provide BCP Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff

/*************************************************************/

NAME:           Send FE Msgs
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A213
DESCRIPTION:    Converts the internal system structures into a
                form that can be transferred to the other
                processors.
INPUTS:
OUTPUTS:        O1, Output or Reply Pkt
                O2, Optimized Query or BCP Cmd Pkt
                O3, Storage Cmd Pkt
                O4, Data or Data Dic Pkt
CONTROLS:       C1, Queue of Outgoing FE Msgs
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A21, Provide Frontend DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Send Shutdown Reply
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A33
DESCRIPTION:    Informs the users that we are closed.
INPUTS:
OUTPUTS:        O1, Shutdown Reply
CONTROLS:       C1, DB Saved
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A3, Shutdown System
RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff



/*****************************************************************/

NAME:           Shutdown System
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A3
DESCRIPTION:    Provides a safe, orderly method of terminating
                the operations of the DBMS.
INPUTS:         I1, Shutdown Cmd
OUTPUTS:        O1, Shutdown Reply
CONTROLS:       C1, System Idle
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A0, Provide Relational DBMS Support

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Startup BCP
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A13
DESCRIPTION:    Causes the BCP to be booted with the BCP
                Operating System.
INPUTS:
OUTPUTS:        O1, Init BCP
CONTROLS:       C1, BCP Startup
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff


/*****************************************************************/

NAME:           Startup Frontend
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A11
DESCRIPTION:    Causes the frontend to be booted with the FE
                Operating Systems.
INPUTS:
OUTPUTS:        O1, Init FE
CONTROLS:       C1, Frontend Startup
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Startup QP
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A15
DESCRIPTION:    Causes the QPs to be booted.
INPUTS:
OUTPUTS:        O1, Init QP
CONTROLS:       C1, QP Startup
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A1, Initialize Database System

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff



/*****************************************************************/

NAME:           Trim Branch
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A22245
DESCRIPTION:    Removes the query job from the task tree.
INPUTS:         I1, Task Tree
OUTPUTS:        O1, Task Tree
                O2, Trimmed Branch
CONTROLS:       C1, QP Stopped
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2224, Manage QP Assignment/Release

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

```
/*****************************************************************/

NAME:           Update Task Tree
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A2226
DESCRIPTION:    Checks for query completion, removes old
                information, and deletes the query from the
                task tree.
INPUTS:         I1, Task Tree
OUTPUTS:        O1, Storage Cmd
                O2, Response
                O3, System Idle
                I1, Task Tree
CONTROLS:       C1, QS Complete
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A222, Execute BCP DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff

/*****************************************************************/

NAME:           Verify Access
TYPE:           Activity
PROJECT:        BCP
NUMBER:         A21222
DESCRIPTION:    Provides data security checks.  It verifies if
                the user has access rights to the relation,
                field, or command.
INPUTS:         I1, Data Dic
OUTPUTS:        O1, Legal DDL Cmd
                O2, Legal Data, BCP Cmd and/or Storage Cmd
                O3, Legal Query or Cmd
CONTROLS:       C1, Correct Query, Data, or Cmd
MECHANISMS:
ALIASES:
PARENT ACTIVITY: A2122, Execute Preliminary DBMS Functions

RELATED REQUIREMENT NUMBER:

VESION:         1.0
DATE:           11/26/84
AUTHOR:         Capt. Dale M. Pontiff
```

Appendix D:

Sample Query in the Frontend

A sample query is shown here to provide a general overview of the actions the FE must take to build a query tree that includes all pertinent data from the Data Dictionary. It assumes all domains are made from basic data types (i.e. integer, float, long integer, double precision float, character string, bit string). It would be possible to also provide max/min range values of each domain element, but this would significantly increase the complexity of the software within the FE and the QPs.

In the sample, the database contains the following domain, and two base relations:

Domain Name	Data Type
name	char (varying)
address	char (varying)
inches	integer
lbs	integer
age	integer
SSAN	integer
sex	char (1)
title	char (varying)
pay	float
skill	char (1)

Personnel Relation

Field Id	Attribute Name	Attribute Domain
1*	Name	name
2	Addr	address
3	Height	inches
4	Weight	lbs
5	Age	age
6	SSAN	SSAN
7	Sex	sex
8	Job_title	title

Job Relations

Field Id	Attribute Name	Attribute Domain
1*	Job	title
2	Pay	pay
3	Skill_level	skill

In this sample, the employer wishes to determine what relation (if any) there is between the employees sex, and pay for everyone under 50 years of age.

```
select sex, age
from personnel, job
where personnel.job_title = job.job
    and personnel.age < 50
```

This query is passed from the host to the FE. The FE checks the syntax, and user access rights, (for the purpose of the demonstration, it is will assumed that this is a valid query). The FE then logs the transaction and optimizes the query into a tree form. Figure D-1 shows the query as an optimized query tree. The query step nodes are numbered in the order they are discussed, not processed.

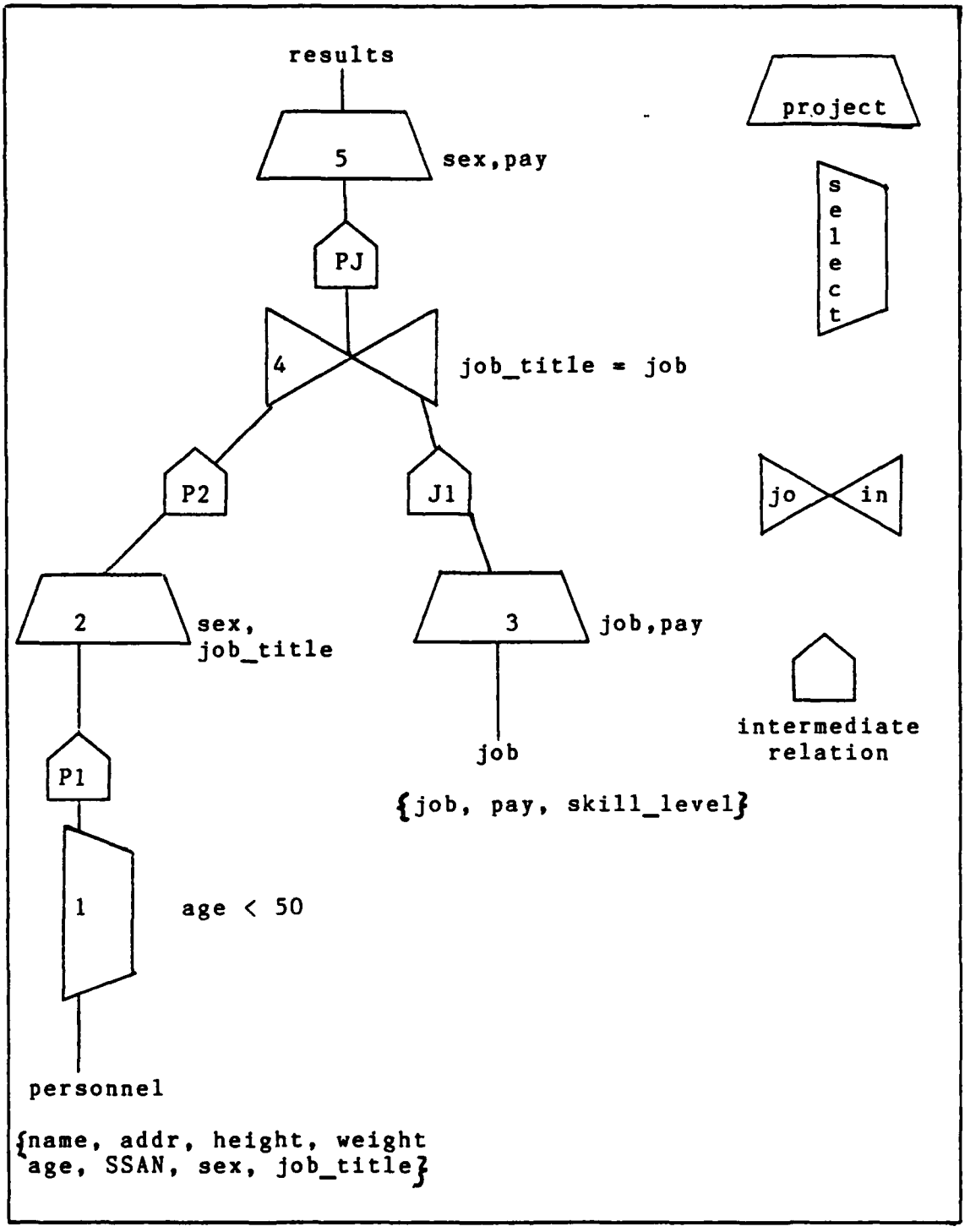


Figure D-1. Optimized Query Tree.

Since the data dictionary is only available to the FE, it must build self contained query steps for the QPs to act on. To do this, it passes pertinent data about each query step down to the QP in thru the selection criterion, attribute list, and modification list.

For node #1, (select personnel tuples where 'age' < 50), the selection criterion is used to tell the QPs to compare the fifth attribute field (age) with a constant value of 50. The resulting relation (P1) has the same format as the input relation (personnel).

Node #2 projects out only the attribute fields sex and job_title (fields 7 and 8). This is done by setting the attribute list in the query step to fields 7 and 8. The resulting relation (P2) consist of only two attribute tuples; {(1, sex) and (2, job_title)}. The ordering is determine by the attribute list. If the attribute list were set to fields 8 and 7, then the resulting relations would contain tuples with {(1, job_title), (2, sex)}.

Since the Select Node (node #1) uses only the selection criterion, and the Project Node (node #2) uses only the attribute list, it is feasible to combine these operation into a single Select/Project query step. This would provide significant savings in terms of paging, at the cost of software complexity within the backend system.

Node #3 is another project node. It tells the QPs to keep only the first two attribute fields of the relation

'job.' Thus, the output relation (J1) contains tuples with {(1, job), (2, pay)}.

The next node (#4) is a join operation. It uses the selection criterion to tell the QPs to join relation 'P2' with relation 'J1' where P2.job_title = J1.job. The selection criterion is set to compare field 2 with field 1 and concatenate the two relations if they are equal. In a join operation, the first field always refers to the first input relation, while the second field refers to the second input relation. Thus we are comparing the second field in relation P2 (P2.job_title) with the first field in relation J1 (J1.job). The resulting relation (PJ) consist of tuples with the following attribute set:

{(1, sex), (2, job_title), (3, job), (4, pay)}

Note that the second relations is concatenated behind the first relation.

The final node (#5) removes the unwanted field in relation PJ giving the final result of {(1, sex), (2, pay)}. Again, since the join operation does not use the attribute list portion of the query step, it is possible to create a combined Join/Project node at the cost of software complexity.

The information passed down to the QPs thru the query steps consists only of the field id (number), not the field name. So, the FE must know what the tuples of each intermediate relation will look like.

Appendix E:

Summary Paper for a Backend Control Processor for a Multi-Processor Relational Database Computer System

Introduction

Work was begun on the Multi-Processor Backend Relational Database Management Computer System in 1981 by Robert Fonden. His purpose was to design a database machine to relieve the main frame computer of the DBMS tasks. This would free up the resources of the main frame for other tasks while supplying faster, cheaper responses to user database queries.

The initial work being done is of an investigative nature. Methods are being tested to determine where major advances to the system can be obtained, but the final system configuration is not fixed.

The current design configuration consists of a frontend processor (FE), a backend control processor (BCP), several query processors (QPs), a fast multi-port memory unit (MBU), and a permanent storage device (MSU) (See Figure E-1).

Overview

The FE is connected to a host system, network, or CRT and receives any incoming user queries or commands. It performs syntax and security checks against the input by referencing the database data dictionary (which is stored on the MSU). If the input was a retrieval query, the FE

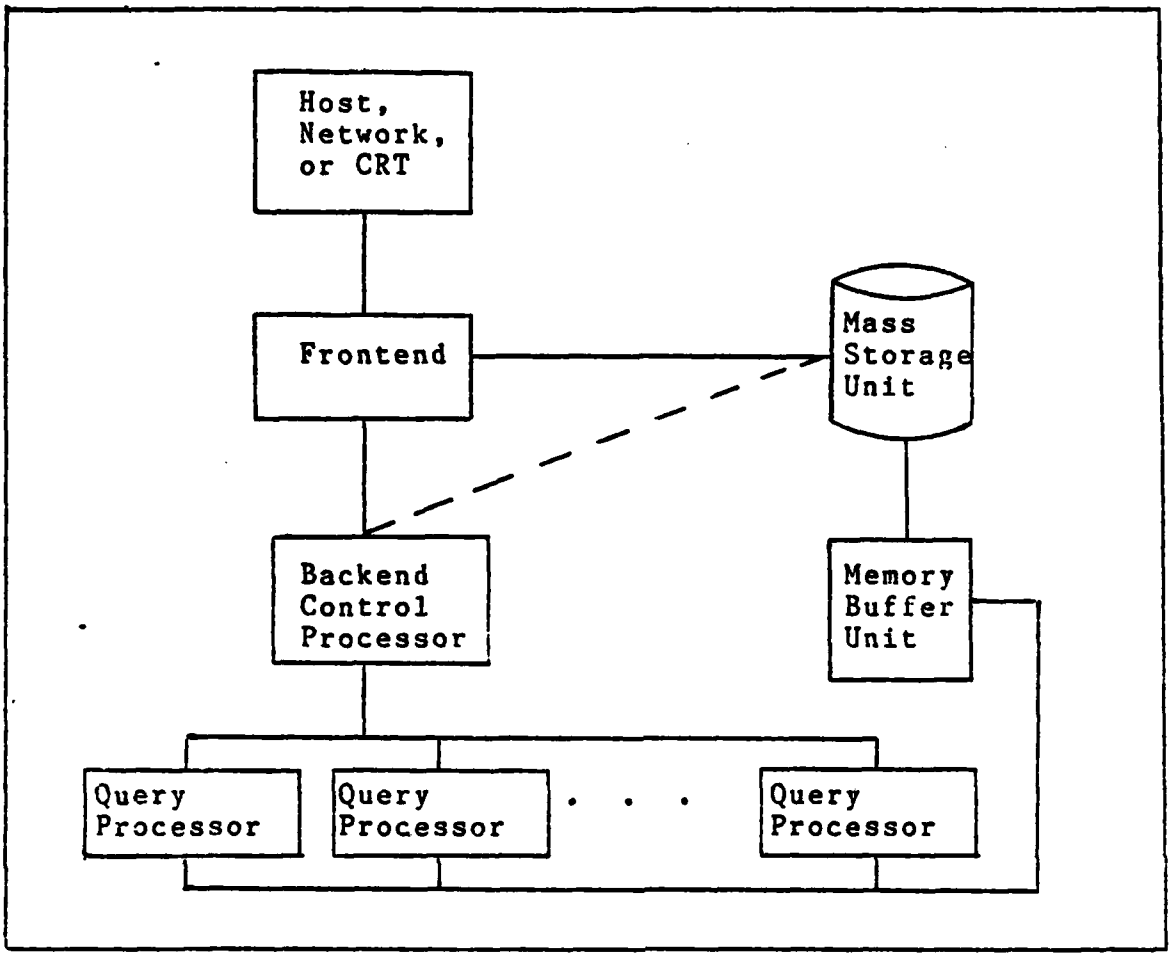


Figure E-1. Physical Design of Backend System

optimizes it, and stores it as a query tree (See Figure E-2). Any commands that modify the data dictionary are executed in the FE, all other commands and queries are passed down to the BCP. Upon the completion of a query/command by the BCP, the FE receives a response message. For update queries and commands, the response contains a reply about the sucess or failure of the query/command. For retrieval queries, the response contains the name of the output relation on the MSU. The FE then transfers any reply or output relation to

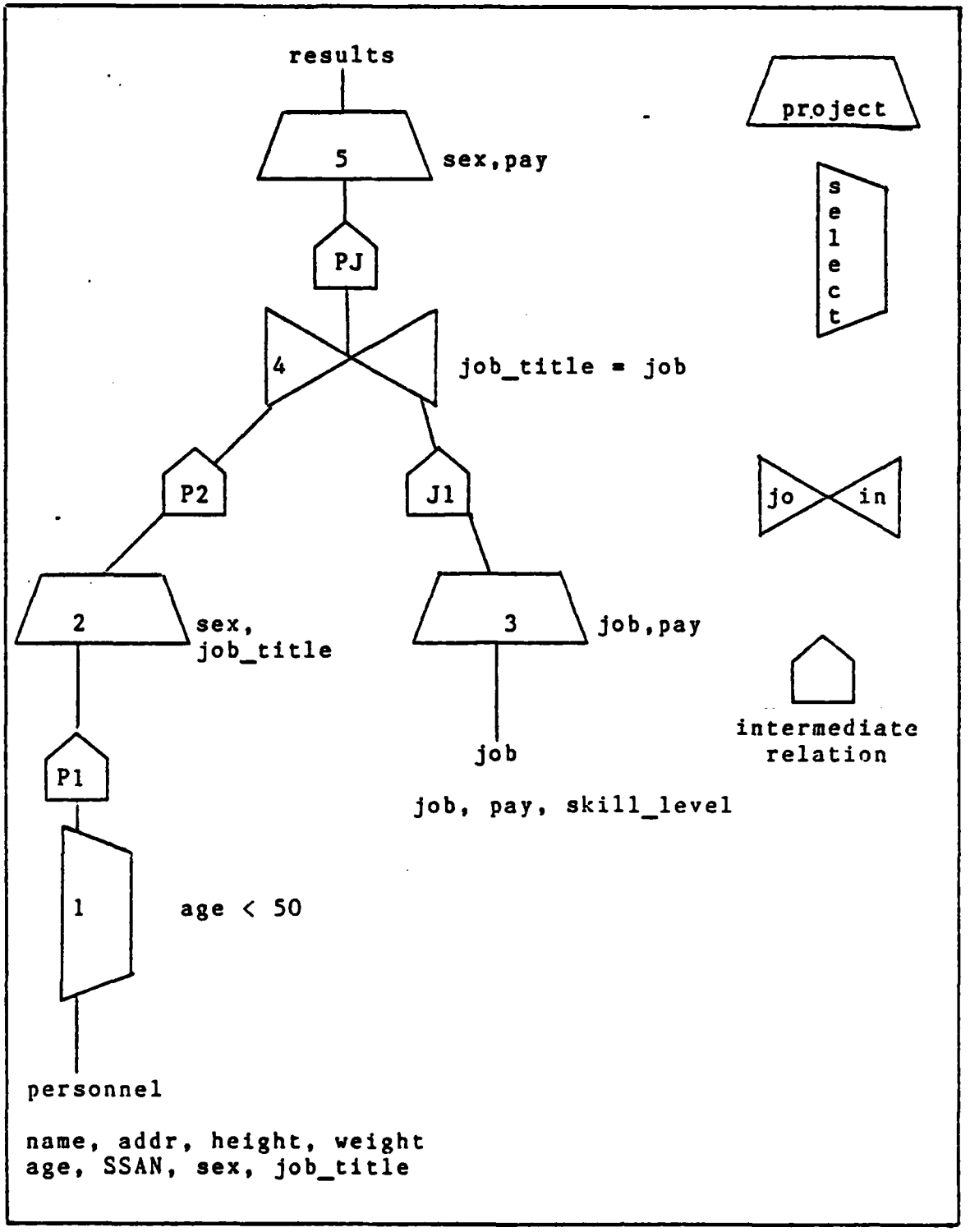


Figure B-2 Optimized Query Tree.

the host system and removes the output relation (if any).

The BCP receives update queries, optimized retrieval queries, and miscellaneous job control commands (stop/restart /abort query, change priority, status, etc.). The BCP's job is to control the assignment of the QPs to specific query steps, manage the system paging algorithms, and control the creation/deletion of temporary relations. It maintains a list of all query steps, the status of each QP and the MBU, and the file size and name of each temporary and base relation it must access.

It is able to direct the MSU to read/write data pages from any file into and out of the MBU, but does not need to exchange any data with the MBU itself (except control messages).

The QPs perform the actual update/retrieval step against the database. Each QP receives a single query step from the BCP along with the necessary paging information to allow them to access the proper page in the MBU (for both input relations and output relations). As a QP completes a page in the MBU, it informs the BCP and begins work on the next page of the relation. The BCP is able to free the page just completed by the QP and direct the MSU to read/write to that page in the MBU. By allowing the QPs to queue up paging information messages, a buffering scheme is achieved. This allows the BCP to send several input/output pages to the QP

and provide smooth, continuous processing by the QPs.

The MSU provides a permanent storage device for the backend system. It stores the database data dictionary and all base relations. It also provides storage space for any temporary files that must be paged out of the MBU.

The MBU provides a fast common scratch pad for the QPs to manipulate the data. It is managed indirectly by the BCP thru the MSU and QPs. The BCP controls which pages the MSU and QPs access and in what fashion (read/write).

The BCP (projected)

The BCP must coordinate the actions of the backend machine. It is connected to the QPs in a master/slave relationship and directs their actions by passing them query steps and paging information. It must be able to control the MSU to get the data pages down to the QPs (thru the MBU). It indirectly controls the MBU by controlling the paging of the MSU and QPs into/out of the MBU. Finally, it must coordinate the recombining of several output relations generated by multiple QPs acting on the same query step.

Upon receiving a query tree from the FE, it determines the priority of the query and adds it to the active query tree (this is a collection of all queries within the backend). Any query step which has some data available to from all of its input files is eligible to be assigned to a QP. Any time the active query tree is not empty, and there is at least one idle QP, the BCP examines the active query tree to

determine which eligible node should be assigned to the QP. This allows the BCP to dynamically decide how many QPs to assign to any eligible query step based on the current work-load, and status of other QPs.

Once the QP is assigned to the query step, the BCP must determine how many pages of storage in the MBU should be dedicated to the query step/query processor pair. It allocates the storage based on the query step operation (select, project, join, etc.), the amount of free stroage in the MBU, and the status of the other QPs in the system. The storage allocation algorithm is essentially a virtual memory manager which guarantees each active QP a minimum number of memory pages based on the query step operation.

As the QPs generate output relations, the BCP must decide whether or not to page the data out to the MSU. If the relations are small, or the output data is being pipe-lined into another QP as input, the BCP will attempt to keep the data in memory. Otherwise, the data must be moved to the MSU to make room for input/output data needed by other QPs. The BCP must be able to create and delete temporary files on the MSU and manage these files (so it knows which file con-tains what output).

If a query step is large enough to warrent the action of two or more query processors, the output must be combined in a manner which can remove duplicate tuples (if required). This is done by breaking the output file(s) into sizes that

will fit into the memory allotted to a QP and performing an in-place sort (i.e. heapsort, or quicksort) on the data. After two portions of the file are sorted, a merge sort algorithm is used to combine them into a single larger file (removing duplicate tuples during the merge). By continuing in this fashion, all duplicate tuples can be eliminated, and the file combined into a single output relation.

Current Implementation of the BCP

The previous section discussed how the BCP should eventually operate, although what is currently implemented falls short in some areas.

The BCP stores incoming queries in a doubly linked circular priority list. Each query is in a modified tree form (update queries contain only a single query step; thus are a trivial tree), with leaf pointers connecting the bottom most leaves of the query tree (See Figure E-3). Only leaf nodes are eligible for assignment to the QPs; thus pipelining is not currently supported.

The QP assignment algorithm does dynamically assign the QPs, but uses a very simple approach. First, it assigns one QP to each eligible query step (by order of priority). If there are more QPs than eligible query steps, it takes the highest priority leaf node and continues to assign the extra QPs to it until the ratio of pages in the relations to QPs is less than a constant value (currently 25). If there are

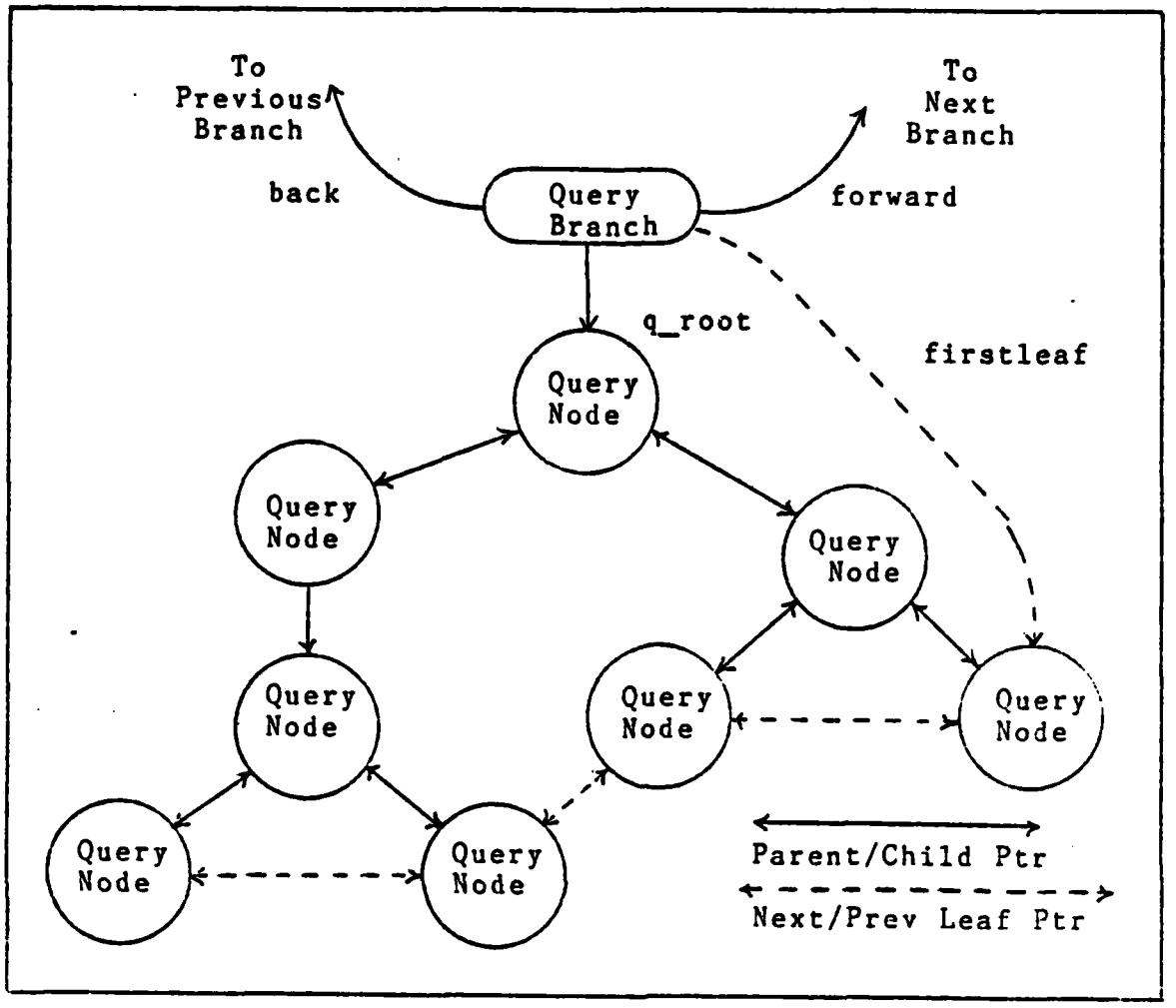


Figure E-3. Query Tree (Branch).

still idle QPs, the process begins again on the next highest leaf. This continues until all the QPs are assigned, or every eligible query step has been checked. Thus, there may be idle QPs because the cost of splitting a node below a certain point is uneconomical (i.e. it takes more time to recombine the output relations).

Because AFIT does not currently have a multi-port memory

(or a suitable substite) for experimental projects, the MBU was eliminated (temporarily) from the implemented version of the backend system. Instead, each QP must reserve memory storage within their own RAM. The storage allocation algorithm on the BCP is limited by this restriction, and cannot dynamically allocate more storage space for any single query step than is available in the QP. This also prevents the BCP from giving one QP more storage than another on a dynamic basis, and eliminates the ability to pipeline data without transferring the data between processors.

The BCP does have a sophisticated file management capability. It maintains an output file for each query step that a QP acts on, and removes any temporary file as soon as possible. Unfortunately, time did not permit the implementation of the sort or merge paging algorithm, so the system is not able to eliminate duplicate tuples at this time.

Conclusion

The development of the Backend Relational DBMS Computer System is still in its infancy. Several areas have been advanced, and some solutions proposed. The ability to dynamically allocate QPs and memory space, and the general system paging algorithm are implemented, but a tremendous amount of work remains to be completed on the BCP and the backend system.

Bibliography

1. Advanced Digital Corporation. Technical Manual for Super Quad. Garden Grove, California: Advanced Digital Corporation.

2. Advanced Digital Corporation. Technical Manual for Super Slave. Garden Grove, California: Advanced Digital Corporation.

3. Boral, Haran and others. "Implemenatation of the Database Machine DIRECT," IEEE Transactions on Software Engineering, Vol SE 8, No 6, November 1982.

4. Date, C. J. An Introduction to Database Systems (Third Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

5. DeWitt, David J. "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computers. Vol C-28, June 1979.

6. Fonden, Robert W. Design and Implementation of a Backend Multiple-Processor Relational Data Base Computer System, Master Thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, 1981.

7. Hartrum, Thomas. Air Force Institute of Technology, Department of Electrical Engineering. Memorandum on 1984 AFIT Thesis Research in Database Management System Design. Wright-Patterson AFB, Ohio, May 1984.

8. Hsiao, David K. and M. Jaishankar Menon. Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I), Contract N00014-75-C-0573 Office of Naval Research. Ohio State University, Columbus, Ohio, July 1981.

9. Hunter, Bruce H. Understanding C. Berkeley, California: SYBEX Incorporation. 1984.

10. Kerr, Douglas S. and others. The Implementation of a Multi-Backend Database System (MDBS): Part I - Software Engineering Strategies and Efforts Toward a Prototype MDBS, Contract N0014-75-C-0573, Office of Naval Research. Ohio State University, Columbus, Ohio, January 1982.

11. Peters, Lawrence J. Software Design Methods and Techniques. New York: Yourdon Press. 1981.

12. Rogers, William R. Development of a Query Processor for a Back-end Multiprocessor Relational Database Computer, Masters Thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, 1981.

13. Roth, Mark A. The Design and Implementation of a Pedagogical Relational Database System, Masters Thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, 1979.

14. Ullman, Jeffrey D. Principles of Database Systems (Second Edition). Rockville, Maryland: Computer Science Press, 1982.

15. Zolman, Lear. BDS C Compiler v1.5 User's Guide. Manual. Brighton, Massachusetts.

VITA

Captain Dale M. Pontiff was born on 14 August 1957 at Westover AFB, Massachusetts. He graduated fron high school in Lafayette, Louisiana, in 1975 and attended the University of Southwestern Louisiana from which he received the degree of Bachelor of Science in Computer Science in May 1979. Upon graduation, he received a commission in the USAF through the ROTC program. He was assigned to the Data Services Center, Pentagon, Washington DC., were he was a system analyist on the Honeywell Multics computer. Since to the School of Enginerring, Air Force Institue of Technology, in May 1983.

Permanent address: 301 Woodvale Ave.
Lafayette, LA 70503

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/ENG/84D-22	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable): RADC/CO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Griffiss AFB, New York 13441	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	

12. PERSONAL AUTHOR(S): Dale M. Pontiff, Capt. USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Thesis	FROM _____ TO _____	84 Dec	227

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	
09	02		Database, Database Machine, Data Management, Backend Processor
			Management Info System, Multiprocessors, Relational Database

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: BACKEND CONTROL PROCESSOR FOR A MULTIPROCESSOR RELATIONAL DATABASE COMPUTER SYSTEM

Thesis Advisor: Dr. Thomas C. Hartrum

Approved for Public Release: IAW AFR 190-17.
1 Feb 85
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED XX SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Thomas C. Hartrum	513-255-3576	AFIT/ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This ~~paper~~ discusses the design and development of a control processo. for a multi-processor relational database machine. The objective was to create the software needed to allow a micro-processor to receive relational query trees from a frontend processor, and to distribute the work load between several other slave processors.

The requirement analysis of the controller determined that the controller must provide three major functions within the backend database machine. It must assign slave processors to query operations, control the system paging, and manage file creation and deletion. Next, the thesis proves that each query operation canbe sucessfully split across several slave processors and the results be recombined to provide the same response as a query executed on a simgle processor. Finally, the thesis gives a detailed description of the software algorithms used by the BCP to manage the backend system.

END

FILMED

4-85

DTIC